UNIVERSIDAD DE SALAMANCA
FACULTAD DE CIENCIAS

PROYECTO DOCENTE PARA EL PERFIL

INGENIERÍA DEL SOFTWARE Y ORIENTACIÓN A OBJETOS

ÁREA DE CONOCIMIENTO
CIENCIA DE LA COMPUTACIÓN E
INTELIGENCIA ARTIFICIAL

Francisco José García Peñalvo

Salamanca, Abril de 2000

DATOS DE LA CONVOCATORIA

CLAVE:	***1999DFCAC6***
CÓDIGO DE LA PLAZA:	***G062/D36220***
FECHA DE LA CONVOCATORIA:	***31 de julio de 1999***
FECHA DE PUBLICACIÓN EN B.O.E:	***11 de septiembre de 1999***
CUERPO DOCENTE:	***Profesores Titulares de Escuela Universitaria***
ÁREA DE CONOCIMIENTO:	***Ciencia de la Computación e Inteligencia Artificial***
DEPARTAMENTO:	***Informática y Automática***
CENTRO:	***Facultad de Ciencias***
UNIVERSIDAD:	***Universidad de Salamanca***
ACTIVIDAD A REALIZAR:	***Docencia en materias de Ingeniería del Software y Orientación a Objetos***

Cita recomendada:

García-Peñalvo, F. J. (2000). *Proyecto Docente Profesor Titular de Escuela Universitaria. Perfil Ingeniería del Software y Orientación a Objetos. Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial*. Salamanca, España: Departamento de Informática y Automática. Universidad de Salamanca. doi:10.5281/zenodo.1015024.

Quisiera dar mi agradecimiento a todos mis antiguos compañeros de la Universidad de Burgos, así como a mis actuales compañeros de la Universidad de Salamanca por el apoyo recibido. Especialmente quiero hacer una mención especial para mis compañeras María N. Moreno García y Ángeles María Moreno Montero con las que he compartido el esfuerzo de realizar este Proyecto Docente, y para Jesús Maudes Raedo por la revisión tan concienzuda que ha llevado de esta memoria. Pero sobretodo quisiera dedicar este trabajo a Mary Cruz por todo el tiempo que la robado y por haberme descargado en estos últimos meses de 'mis labores domésticas'.

Tabla de Contenidos

Capítulo 1
Introducción

La presente memoria constituye el Proyecto Docente que se ajusta a la plaza de Profesor Titular de Escuela Universitaria, con clave G062/D36220, convocada por resolución rectoral de la Universidad de Salamanca el 31 de julio de 1999 y publicada en el Boletín Oficial del Estado con fecha 11 de septiembre de 1999. Esta plaza se adscribe al Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial que forma parte, en esta Universidad, del Departamento de Informática y Automática.

El perfil de la plaza establece que la actividad a realizar consiste en la docencia en materias de Ingeniería del Software y Orientación a Objeto dentro de la titulación de **Ingeniería en Informática de Sistemas** de la Universidad de Salamanca.

La legislación vigente que regula la provisión de plazas a los cuerpos docentes universitarios, artículo 37 de la L.R.U. de 26 de julio de 1983 y los Reales Decretos de 26 de junio de 1984 y de 13 de junio de 1986, no establece normativa alguna que indique cuál debe ser el contenido del Proyecto Docente. En el que aquí se desarrolla, se presenta, desde una perspectiva global, tanto la concepción actual de la Ingeniería del Software y la Orientación a Objeto, como la metodología docente que permita la capacitación de los alumnos para la realización de las tareas profesionales y/o científicas que deberán desarrollar.

La estructura que toma este documento se deriva de la consulta de diversos Proyectos Docentes presentados en otras tantas oposiciones a plazas de los cuerpos docentes universitarios [Vivaracho, 1996], [Zazo, 1996], [Marqués, 1998], [Maudes, 1998], [Moreno, 1999], [Rodríguez, 1999], [Pelechano, 2000], [Moreno, 2000]. Así, el resto de la memoria se organiza según se expone a continuación.

En el capítulo dos se hace una breve reflexión sobre las funciones de la Institución Universitaria y se presenta el marco curricular en que se desarrollan las mismas en la

Ingeniería Técnica en Informática de Sistemas, que se imparte en la Facultad de Ciencias de la Universidad de Salamanca.

El capítulo tres efectúa una valoración crítica de los distintos métodos de enseñanza, instrumentos pedagógicos y sistemas de evaluación, concluyendo la conveniencia de la utilización de aquéllos que se estiman más adecuados para el ejercicio de la docencia en los estudios de la Ingeniería en Informática. En este capítulo se abordan temas que complementan la labor docente: *la investigación* como fuente necesaria de nuevos conocimientos en un área en constante evolución como es la Informática y *el aseguramiento de la calidad en la docencia* mediante la realización de planes de calidad.

En el cuarto capítulo se define el perfil que rige el presente Proyecto Docente, analizando su presencia en la Ingeniería Técnica en Informática de Sistemas, los aspectos formativos de la plaza objeto de concurso y las interrelaciones existentes con otras asignaturas de la misma titulación, así como de asignaturas de la Ingeniería Informática (estudios de Segundo Ciclo). Para cerrar este capítulo se hace un estudio del perfil de la plaza tanto a escala internacional, analizando las propuestas curriculares existentes sobre la formación en Informática, como en el ámbito nacional recorriendo diferentes planes de estudios de las Ingenierías Informática (tanto técnicas como superiores) impartidas por universidades españolas.

El capítulo cinco constituye el bloque central del Proyecto Docente. Presenta y justifica el temario de las asignaturas Ingeniería del Software y Programación Orientada a Objetos que, dentro de plan de estudios vigente, recogen los conceptos teórico-prácticos del perfil de la plaza. Se concluye este capítulo con la enumeración de diferentes recursos (*revistas, congresos, sitios web, grupos de noticias*) que pueden ser útiles en el desarrollo de las asignaturas objeto del presente Proyecto Docente.

La memoria se completa con una serie de apéndices. El apéndice A recoge de forma íntegra el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca. Los apéndices B y C presentan un glosario con los acrónimos y los términos respectivamente más utilizados en el área en la que se centra el Proyecto Docente.

Cabe reseñar que para la realización del presente Proyecto Docente se ha seguido un *proceso iterativo e incremental,* que ha permitido ir elaborando cada uno de los capítulos de forma progresiva, incorporando en cada iteración mayor información, lo que se asemeja a la utilización del modelo de ciclo de vida en espiral definido por **Barry Boehm** [Boehm, 1988] a la documentación, pero aplicado a cada capítulo de forma independiente, e incluso, como en el caso del capítulo cinco, aplicado a la descripción detallada de cada uno de los temas de las dos asignaturas que conforman este informe, hecho que es más acorde con el modelo fuente de ciclo de vida definido por **Henderson-Sellers** y **Edwards** [Henderson-Sellers and Edwards, 1990]. Así, se

tiene una aplicación de los modelos de ciclo de vida orientados a objetos en otros contextos diferentes al del desarrollo del software, tal y como se desprende de [Concepcion, 1998].

Por último, señalar que para la elaboración de este Proyecto Docente, además de la experiencia adquirida en mi actividad docente, la reflexión y el estudio personal, también han constituido una valiosa aportación las opiniones y apoyo de muchas personas, entre ellas mis compañeros; a ellas, por tanto, manifiesto mi más sincero agradecimiento.

1.1 Referencias

[Boehm, 1988] Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

[Concepcion, 1998] Concepcion, Arturo I. "*Using an Object-Oriented Software Life-Cycle Model in the Software Engineering Course*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 30-34. 1998.

[Henderson-Sellers and Edwards, 1990] Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

[Marqués, 1998] Marqués Corral, José Manuel. "*Proyecto Docente e Investigador. Ingeniería del Software e Inteligencia Artificial*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Octubre, 1998.

[Maudes, 1998] Maudes Raedo, Jesús Manuel. "*Proyecto Docente para las Asignaturas: Sistemas de Gestión de Bases de Datos y Administración de Bases de Datos*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Escuela Universitaria Politécnica. Universidad de Burgos. Noviembre, 1998.

[Moreno, 2000] Moreno Montero, Ángeles María. "*Proyecto Docente. Redes de Ordenadores y Bases de Datos*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Abril, 2000.

[Moreno, 1999] Moreno Rodilla, Vidal. "*Proyecto Docente e Investigador. Control e Instrumentación de Procesos Químicos*". Área de Conocimiento de Ingeniería de Sistemas y Automática. Facultad de Ciencias. Universidad de Salamanca. Junio, 1999.

[Pelechano, 2000] Pelechano Ferragud, Vicente. "*Proyecto Docente. Ingeniería del Software*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Valencia. Marzo, 2000.

[Rodríguez, 1999] Rodríguez Rubio, Miguel. "*Proyecto Docente. Análisis de Sistemas Informáticos*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Facultad de Informática. Universidad de A Coruña. Julio, 1999.

[Vivaracho, 1996] Vivaracho Pascual, Carlos Enrique. "*Proyecto Docente. Teoría de la Información y Codificación – Sistemas Operativos*". Área de Conocimiento de Ciencias de

la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Julio, 1996.

[Zazo, 1996] Zazo Rodríguez, Ángel Francisco. "*Proyecto Docente para el Perfil Teleinformática Aplicada a las Ciencias de la Documentación*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Facultad de Traducción y Documentación. Universidad de Salamanca. Diciembre, 1996.

Capítulo 2
Ámbito Académico

Todo proyecto docente debe tener en cuenta las características contextuales en el que se va a desarrollar para poder adaptarse a ellas. Cada Universidad, cada Centro y cada Titulación presenta una serie de particularidades que el docente debe conocer y considerar a la hora de desarrollar su acción formativa. Por ello se muestra a continuación una panorámica de este entorno docente. En primer lugar se presenta el concepto de Universidad, sus objetivos y sus funciones, como marco de todas las actividades que en ella se realizan, y en particular de la que nos ocupa. A continuación se concretan las atribuciones y peculiaridades de la Facultad, para posteriormente, descendiendo un peldaño más en la estructura universitaria, comentar las características del Departamento y del Área de Conocimiento. Tras esta revisión de la estructura universitaria se procede a presentar las características y particularidades del alumnado, así como la infraestructura disponible. Con el análisis del marco curricular de las Titulaciones de Ingeniería Técnica en Informática y de Ingeniería en Informática, se cierra esta panorámica que define de forma específica el marco educativo en el que se encuadra la labor docente objeto del presente estudio.

2.1 La Universidad

2.1.1 Concepto y misión

Para poder concretar el trabajo dentro de una institución tan compleja como es la Universidad, primero es preciso definir su misión. Así, valorando con posterioridad la realidad actual puede plantearse la necesaria reforma desde lo que es hacia lo que debiera ser.

La Universidad ha desempeñado y desempeña un papel clave en el desarrollo y avance de la sociedad, por tanto debe tener un cuidado especial al llevar a cabo sus

actividades. Su ***finalidad*** en el ámbito educativo no es sólo proporcionar información, sino *conseguir una formación integral de los estudiantes de cara al desempeño eficaz de su futura actividad profesional*. Su actividad, así como su autonomía, se fundamentan en el principio de la libertad académica, que se manifiesta en las libertades de cátedra, de investigación y de estudio.

A pesar de su larga data, las ***funciones básicas*** de la enseñanza universitaria, tal y como se concibe en la actualidad, fueron establecidas por **José Ortega y Gasset** en su trabajo *Misión de la Universidad* [Ortega, 1982]. Estas funciones son:

- Transmisión de cultura, entendida ésta como sistema de ideas vivas de una época.
- Preparación para el ejercicio profesional.
- Realización de investigación científica y educación de nuevos hombres de ciencia.

Estas funciones de la Universidad propuestas por Ortega y Gasset, no sólo no han perdido vigencia, sino que incluso se encuentran recogidas en la Ley Orgánica 11/1983 de 25 de agosto de Reforma Universitaria [MEC, 1983], al afirmar en su preámbulo: "*El desarrollo científico-técnico, la formación profesional y la extensión de la cultura son las tres funciones básicas que de cara al siglo XXI debe cumplir esa vieja y hoy renovada Institución Social que es la Universidad española*".

Así, en el Título preliminar, artículo primero, se dice textualmente que:

1. *El servicio público de la educación superior corresponde a la Universidad, que lo realiza mediante la docencia, el estudio y la investigación.*
2. *Son funciones de la Universidad al Servicio de la sociedad:*
    a. *La creación, desarrollo, transmisión y crítica de la ciencia, de la técnica y de la cultura.*
    b. *La preparación para el ejercicio de actividades profesionales que exijan la aplicación de conocimientos y métodos científicos o para la creación artística.*
    c. *El apoyo científico y técnico al desarrollo cultural, social y económico, tanto Nacional como de las Comunidades Autónomas.*

Por tanto la Universidad tiene una labor docente y otra investigadora. Ambas actividades se complementan mutuamente: la investigación permite la ampliación de conocimientos y por consiguiente la actualización docente, y a su vez la docencia hace posible la sedimentación y la difusión de los resultados de la labor investigadora.

Centrando la atención en el caso concreto de la ***Universidad de Salamanca***, los fines particulares, establecidos el artículo segundo de sus Estatutos[1], son los siguientes:

a) La ampliación del conocimiento por medio de la investigación en todas las ramas de la cultura, la ciencia y la técnica.

b) La transmisión crítica del saber a través de la actividad docente y de la discusión científica.

c) La contribución a la formación y perfeccionamiento de profesionales cualificados.

d) La promoción, enaltecimiento y difusión de la lengua española.

e) La contribución a la mejora del sistema educativo.

f) El fomento y expansión de la cultura, mediante la acogida y estímulo de la actividad intelectual en todos los ámbitos.

g) El asesoramiento científico, técnico y cultural a la sociedad, dirigido a la satisfacción de sus necesidades.

h) La cooperación al desarrollo científico y técnico, cultural y social de Castilla y León, de España y de todos los pueblos.

La enseñanza universitaria no se reduce por tanto a la formación técnica del estudiante, sino que debe perseguir el carácter educacional del aprendizaje. Los alumnos deben beneficiarse de una educación que les permita desarrollar al máximo sus aptitudes y su potencial creativo. Como se desprende de sus Estatutos, es objetivo primordial e ineludible de la Universidad de Salamanca lograr que la enseñanza que en ella se alcance el más alto grado de calidad, y tienda a la formación integral y crítica de sus miembros. Promueve, como uno de los fines fundamentales de su actividad, la formación de investigadores, así como el fomento y coordinación de la investigación científica y técnica.

2.1.1.1 La Universidad y la cultura

El concepto de Cultura de Ortega y Gasset, se entiende repasando someramente la propia historia de la Universidad.

En la Edad Media la Universidad no formaba profesionales ni científicos. Tan sólo era el lugar donde se trasmitía el sistema de ideas sobre el mundo y la humanidad que entonces poseía el hombre, llegando a crearse un repertorio de convicciones que había de dirigir definitivamente la existencia de los hombres que lo adquirían. El universitario

[1] Recogidos en el Boletín Oficial del Estado de 5 de julio de 1988: Real Decreto 678/1988 de 1 de julio. Posteriormente en el Boletín Oficial del Estado de 10 de agosto de 1991; Real Decreto 1292/1991 de 2 de agosto, se aprueba la modificación de los estatutos de la Universidad de Salamanca.

trabajaba para conseguir ideas claras y firmes sobre el Universo, convicciones positivas sobre lo que son las cosas y el mundo.

No es hasta finales del siglo pasado, donde por razones de abarcabilidad de conocimientos, cada vez más técnicos y complejos, se acaba imponiendo la especialización o separación entre lo que se denominó cultura o malentendido humanismo y ciencia. Pero según **F. Savater** "*las facultades que el humanismo pretende desarrollar son la capacidad crítica de análisis, la curiosidad que no respeta dogmas ni ocultamientos, el sentido de razonamiento lógico, la sensibilidad para apreciar las más altas realizaciones del espíritu humano, la visión de conjunto ante el panorama del saber, etc*" [Savater, 1998], por lo que repitiendo sus palabras: "*Francamente, no conozco ningún argumento serio para probar que el estudio del latín y el griego favorecen más estas deseables cualidades que el de las matemáticas o la química*" [Savater, 1998] o su consecuencia: el humanismo o cultura no va ligado a la condición práctica o técnica de la disciplina a impartir sino a su servicio a la interpretación del mundo. El hombre no puede vivir sin reaccionar ante su entorno, sin interpretarlo de alguna forma y sin plantearse una conducta a seguir. Esto supone, en palabras de Ortega [Ortega, 1982] "*esa terrible faena de sostenerse en el Universo, de conducirse por entre las cosas y seres del mundo*". "*Cultura es el sistema de ideas vivas que cada tiempo posee. El sistema de ideas desde las cuales el tiempo se vive*".

Pero la casi totalidad de estas convicciones no se las fabrica el individuo, sino que las recibe de su medio histórico, de su tiempo. Aunque siempre coexisten sistemas ideológicos muy diferentes, hay uno que representa el nivel superior del tiempo, un sistema plenamente actual y ese es la Cultura.

En nuestra época el contenido de la cultura viene en su mayor parte de la ciencia. Pero la cultura hace con la ciencia lo mismo que con la profesión: saca de ellas lo vitalmente necesario para interpretar la existencia. La cultura necesita poseer una idea completa del hombre y del mundo, y no se detiene como la ciencia, allí donde terminan los métodos de absoluto rigor científico. Además, como advierte B. Russell [Russell, 1997] "*Una dictadura de hombres de ciencia sería muy pronto horrible. La habilidad sin sabiduría puede ser puramente destructiva y es muy probable que se revelara así. Aunque sólo sea por esta razón, es de gran importancia que quienes reciben una educación científica no se limiten a ser científicos, sino que posean cierta comprensión de esa clase de sabiduría que sólo puede impartir, si es posible tal cosa, la vertiente cultural de la educación*".

La Universidad, no debe, por tanto, convertirse en un lugar donde sólo se imparten conocimientos profesionales y se enseñe a hacer Ciencia. En ella se trata de conseguir el máximo desarrollo de los estudiantes universitarios, ayudándoles a descubrir sus cualidades, sus capacidades, su planteamiento vital; a ser capaces de comunicarse, de hacer crítica y autocrítica, de adquirir una postura ante su profesión, ante la ciencia, la técnica, la vida, ante el mundo. En definitiva, de adquirir ese sistema vital de ideas que

les ayude a encontrar más honestamente su función social y su compromiso ante el mundo y ante ellos mismos. Se trata en suma de conseguir, parafraseando a Whitehead "*cabezas bien hechas y no sólo bien llenas* ". Por ello, como indicaba Russell [Russell, 1997] los buenos profesores universitarios "*Deberían dar ejemplo del valor del intelecto y la búsqueda del conocimiento. Deberían dejar claro que lo que en cualquier época pasa por conocimiento puede en realidad ser erróneo. Deberían inculcar un temple de búsqueda continua y no de cómoda certeza. Deberían proponerse que sus alumnos fuesen conscientes del mundo como una totalidad... Mediante el reconocimiento de la probabilidad de error, deberían dejar clara la importancia de la tolerancia. Deberían recordar al alumno que aquellos a quienes honra la posteridad muy a menudo han sido impopulares en su época y que, a este respecto, tener el valor de enfrentarse a la sociedad es una virtud de suprema importancia*".

Sin embargo, fácilmente se observa que la Universidad actual no cumple esta misión, al menos desde un punto de vista formal y organizado. La enseñanza universitaria se ve actualmente desbordada por la cantidad de información que la Ciencia aporta a la enseñanza profesional, y por la propia investigación, usurpando todas estas materias el lugar de transmisión de la Cultura. Esto hace que el producto de la Universidad no sea un hombre centrado en su tiempo. Este problema ya había sido predicho por Ortega y Gasset y descrito en el año 1930. En sus propias palabras [Ortega, 1982], este fallo "*... produce un ser inculto. Un personaje medio que es el nuevo bárbaro retrasado con respecto a su época; arcaico y primitivo en comparación con la terrible actualidad y fecha de sus conocimientos. Este nuevo bárbaro es principalmente el profesional más sabio que nunca, pero también más inculto*".

Por eso es ineludible impulsar de nuevo en la Universidad la "*enseñanza de la Cultura o sistema de ideas vivas que cada tiempo posee*". Es una tarea universitaria imprescindible y como tal recogida en la Ley de Reforma Universitaria [MEC, 1983], que textualmente afirma también en su preámbulo: "*La Ciencia y la Cultura son la mejor herencia que las generaciones adultas pueden ofrecer a los jóvenes y la mayor riqueza que una nación puede generar, sin duda, la única riqueza que vale la pena acumular*".

Sin embargo, es fácil formular este objetivo, y difícil concretar los medios para alcanzarlo en la estructura universitaria actual. Habría que empezar por convencernos a nosotros mismos y a los estudiantes de que no todas las actividades que se salen de la enseñanza estricta de la profesión son una pérdida de tiempo, siendo deseables la transmisión del gusto por la reflexión y la crítica constructiva sobre el entorno social que rodea a una persona y lo que ella misma, desde su ejercicio profesional puede aportar a ese contexto.

2.1.1.2 La Universidad y la formación de profesionales

No sólo la Ley de Reforma Universitaria, como ya se mencionó, recoge entre los fines de la Universidad la formación de los profesionales, sino que los propios Estatutos de la Universidad de Salamanca establecen como fin de la misma "*La contribución a la formación y perfeccionamiento de profesionales cualificados*".

Con anterioridad a cualquier planteamiento sobre la formación profesional, es preciso diferenciar claramente lo que es profesión de lo que es ciencia.

La Ciencia estudia los fenómenos, profundiza en sus causas y en sus mecanismos desarticulándolos con la finalidad de entenderlos, para posteriormente reestructurarlos aportando algo nuevo al conocimiento anterior. El conjunto de estos conocimientos, ordenados, en torno a una materia concreta, y puestos en práctica, es lo que se puede llamar Profesión; o viceversa, la Profesión es la puesta en práctica de los conocimientos obtenidos gracias a la Ciencia. De este modo como afirma Russell [Russell, 1997], el profesional desea cambiar la naturaleza mientras que el científico desea comprenderla y ninguna de las dos actitudes por si solas son útiles al hombre.

La Ciencia pues, es la fuente de los conocimientos profesionales, y al entrar en la Profesión, debe desarticularse como tal, y reorganizarse para ser puesta en práctica. Así, puede resumirse que el investigador "*hace ciencia*", el profesional "*pone en práctica los descubrimientos científicos*" y el docente "*transmite estos conocimientos*".

Acerca de esta diferente función de la Universidad, Ortega y Gasset dice lo siguiente [Ortega, 1982]: "*Es preciso separar la enseñanza profesional de la investigación científica, y que ni en los profesores ni en los alumnos se confunda lo uno con lo otro, so pena de que lo uno dañe a lo otro. En general, el estudiante normal, no es un aprendiz de científico. El Cientifismo en los profesores puede interferir en la formación de profesionales, ya que lo que se conseguirá será producir profesionales mediocres con un mínimo atisbo de ciencia. Es preciso pues sacudir bien de ciencia el árbol de las profesiones, a fin de que quede en ella lo estrictamente necesario, y pueda atenderse a las profesiones mismas*".

En el campo de la Informática la rápida evolución tecnológica implica que la formación no pueda limitarse a impartir los programas encaminados a la obtención del correspondiente título que certifique la capacitación para el ejercicio profesional en sí. Si de verdad se quiere dar respuesta a las necesidades reales de formación que la sociedad plantea, la docencia no puede restringirse a los estudiantes de primer, segundo o tercer ciclo. La Universidad como lugar donde "*se hace Ciencia*", debe estar abierta a cualquier profesional que desee una actualización o ampliación de sus conocimientos, "*enseñando los resultados de esa ciencia*" en cada momento, a través de programas de formación continuada en permanente renovación.

Para no correr el riesgo de no formar ni buenos profesionales, ni buenos científicos, es necesaria la utilización de una metodología adecuada que obligue a detallar qué se

pretende y qué medios se utilizarán para su consecución. Este soporte indispensable que constituye la metodología docente se tratará ampliamente en el capítulo siguiente, por lo que aunque se reconozca como fundamental esta función universitaria, no se desarrolla más bajo este epígrafe.

2.1.1.3 La Universidad y la investigación científica

La Ciencia se puede definir como un conjunto sistemático de conocimientos sobre la realidad observable, fruto de la investigación científica. No es otra cosa que el resultado de la investigación realizada de acuerdo con el método científico, por lo que la investigación científica es la fuente de la ciencia.

Investigación deriva etimológicamente del latín "***in***" (*en, hacia*) y "***vestigium***" (***huella, pista***); su significado es "*hacia la pista*", averiguar algo siguiendo un rastro. Según esto, toda investigación, incluso la científica, es la búsqueda del conocimiento de algo no conocido, o la búsqueda de la solución de un problema. Por ello, los diferentes tipos de investigación no se pueden distinguir en su origen, sino que se van a distinguir por el método. De este modo para que una investigación se considere científica, es necesario que se utilice en ella el método científico.

Según **R. Sierra Bravo** [Sierra, 1986], "*El método científico, consiste en formular cuestiones o problemas sobre la realidad del mundo y de los hombres, sobre la base de la observación de la realidad y la teoría ya existentes; en anticipar soluciones a estos problemas, y en contrastar, con la misma realidad, dichas soluciones o hipótesis mediante la observación de los hechos, su clasificación y su análisis*".

Asimismo Russell [Russell, 1987] afirma que "*para llegar a establecer una ley científica existen tres etapas principales: la primera consiste en observar los hechos significativos; la segunda, en sentar hipótesis que si son verdaderas expliquen aquellos hechos; la tercera en deducir de estas hipótesis consecuencias que pueden ser puestas a prueba por la observación*". Puntualiza, además, que, "*Decir que un hecho es significativo, en ciencia, es decir, que ayuda a establecer o refutar alguna Ley General; pues la Ciencia, aunque arranca de la observación de lo particular, no está ligada esencialmente a lo particular, sino a lo general*".

El desarrollo del método científico puede resumirse en las siguientes etapas:

1. *Planteamiento del problema: reconocimiento de los hechos y reducción del problema a su núcleo significativo.*
2. *Construcción de un modelo teórico a partir de las hipótesis, y su traducción matemática si es posible.*
3. *Deducción de consecuencias particulares.*

4. *Prueba de las hipótesis, que en Informática al tratar con objetos materiales, se realiza por verificación, a diferencia de las ciencias formales cuyo criterio de verdad es la consistencia de los enunciados.*
5. *Introducción de las conclusiones en la teoría, con el consiguiente reajuste del modelo y la aportación de sugerencias para desarrollos posteriores. Esto es, discusión, conclusiones y generalización.*

Definido así el proceso metodológico científico, puede asumirse la definición de Sierra Bravo, que considera la investigación científica [Sierra, 1986] "*como una actividad compleja, inteligente, constituida fundamentalmente por la previa documentación existente, por el proceso de aplicación del método científico a problemas concretos en un área de la realidad observable, buscando respuesta a estos problemas para obtener nuevos conocimientos que se ajusten lo más posible a la realidad investigada, con elaboración de los resultados obtenidos en un trabajo de investigación o tesis y su presentación de forma escrita*".

El hecho de que la Universidad, en la actualidad, no es el único lugar donde se realiza investigación científica es un hecho incuestionable. Pero también lo es, según apunta la Ley de Reforma Universitaria, el hecho de que a pesar de sus limitaciones, la Institución Social mejor preparada para asumir hoy el reto del desarrollo científico-técnico e impulsar la mentalidad y el espíritu científico en España es la Universidad.

2.1.2 La Facultad de Ciencias

Tal y como se recoge en el artículo noveno de la Ley de Reforma Universitaria [MEC, 1983], las ***facultades*** son:

> **"... los órganos encargados de la gestión administrativa y de la organización de las enseñanzas universitarias conducentes a la obtención de títulos académicos"**

En la Universidad de Salamanca las Facultades se rigen por los Estatutos publicados en el B.O.E. de 5 de julio de 1988. Sus tareas principales quedan establecidas en el artículo 16, en el que se le atribuyen:

a) *La elaboración de sus planes de estudio.*

b) *La organización y coordinación de las actividades docentes, así como la gestión de los servicios y medios de apoyo a la investigación y la enseñanza.*

c) *La organización de las relaciones entre Departamentos y con otros Centros, a fin de asegurar la coordinación de la enseñanza y la racionalización de la gestión académica y administrativa.*

d) *La expedición de certificados académicos y la tramitación de propuestas de convalidación, traslado de expedientes, matriculación y otras funciones similares.*

e) *La administración de su presupuesto.*

Dentro de la Universidad de Salamanca, la facultad que se hace cargo de los estudios de Informática es la Facultad de Ciencias. Las titulaciones que imparte son:

- ***Ingeniero Técnico en Informática de Sistemas***
- ***Ingeniero en Informática***
- Licenciado en Ciencias Físicas
- Licenciado en Ciencias Geológicas
- Licenciado en Matemáticas
- Diplomado en Estadística

2.1.3 El Departamento de Informática y Automática

Con la entrada en vigor de la Ley de Reforma Universitaria se produce un cambio en la estructuración y organización de las enseñanzas universitarias. Se establece el Departamento como unidad básica de la estructura universitaria. Concretamente el artículo octavo del título primero establece que los departamentos son:

> **"... los órganos básicos encargados de organizar y desarrollar la investigación y las enseñanzas propias de su respectiva área de conocimiento en uno o varios centros"**

En la misma línea, los Estatutos de la Universidad de Salamanca, en el capítulo primero del título segundo señalan que:

> **"... los Departamentos son las unidades fundamentales de enseñanza e investigación de la Universidad"**

Los Departamentos gozan de autonomía en la gestión y utilización de sus recursos personales y materiales; a ellos corresponde organizar y desarrollar la investigación y las enseñanzas de sus áreas de conocimiento en los diferentes centros en los que tiene asignada docencia. Más detalladamente, las funciones que los Estatutos de esta Universidad asignan a los Departamentos se establecen en el artículo catorce, y son las siguientes:

a) Confeccionar programas e impartir docencia en las áreas de conocimiento de su competencia, bajo la coordinación de los centros afectados.

b) Programar y realizar proyectos de investigación.

c) Planificar e impartir cursos de especialización, perfeccionamiento y actualización de los conocimientos científicos de los titulados universitarios y de sus propios miembros.

d) Promover la participación y el asesoramiento en trabajos de carácter científico, técnico o artístico.

e) Programar e impartir los cursos de doctorado, así como coordinar la elaboración y dirección de tesis doctorales.

f) Fomentar programas de enseñanzas e investigación interdisciplinares e interdepartamentales.

g) Organizar y llevar a cabo investigaciones acordadas en contratos suscritos con personas físicas, entidades públicas o privadas, nacionales o extranjeras.

La estructura universitaria organiza a los docentes e investigadores en Departamentos. Por tanto, para enmarcar las tareas de un profesor universitario se hace necesario la descripción de la unidad a la que pertenece. Así, el Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial está adscrita en la Universidad de Salamanca al Departamento de Informática y Automática. La historia del Departamento es reciente, pues su andadura se inicia por resolución de la Junta de Gobierno de esta Universidad el 31 de Octubre de 1996.

En sus inicios, estaba constituido por dos áreas de conocimiento:

- ***Ingeniería de Sistemas y Automática***
- ***Lenguajes y Sistemas Informáticos***

Posteriormente en 1996, se incorpora el Área de Conocimiento de ***Ciencia de la Computación e Inteligencia Artificial***. Finalmente, en 1998, se crea el Área de Conocimiento de ***Arquitectura y Tecnología de los Computadores***.

La creación y evolución del Departamento viene marcada por la aparición de nuevas titulaciones en la Universidad de Salamanca (*Tabla 2.1*). Así, hasta finales de los años 80, el área Ingeniería de Sistemas centraba su actividad docente en la impartición de asignaturas de en la titulación de Ciencias Físicas. Es a partir de 1988 cuando comienza a impartir asignaturas relacionadas con la Informática en dos nuevas titulaciones: La Diplomaturas de Biblioteconomía y la Diplomatura de Informática.

De esta forma, al comienzo de la década de los 90 comienzan a impartirse en esta Universidad nuevas titulaciones siguiendo la misma tendencia que en el resto de las universidades españolas. Ello da lugar a que aparezca en 1990 el Área de Lenguajes y Sistemas Informáticos intentando satisfacer en un mejor grado las nuevas necesidades docentes e investigadoras de la Universidad de Salamanca. El área nueva, y la ya existente en menor grado, aunque de forma también relevante, tienen un importante

incremento de número de personal y carga docente, donde cada año aparecen nuevas asignaturas en diferentes centros. De esta forma, un área a la que pertenecían tres personas en 1988 se convierte en un Departamento cuya plantilla se puede observar en la *Tabla 2*. El Departamento de Informática y Automática se completa con la creación del Área de Ciencias de la Computación e Inteligencia Artificial en 1996 y del área de Arquitectura y Tecnología de los Computadores en 1998.

CURSO	TITULACIÓN
1987/1988	*Diplomatura en Biblioteconomía y Documentación*
1989/1990	*Ingeniero Técnico Industrial. Esp. Electrónica (Intensificación Electrónica)*
1989/1990	*Diplomatura de Informática*
1992/1993	*Licenciatura en Traducción e Interpretación*
1993/1994	*Ingeniería Química*
1998/1999	*2º ciclo Ingeniero Industrial*
1998/1999	*Ingeniero en Informática (2º ciclo)*

Tabla 2.1. Año de creación de nuevas titulaciones técnicas en la Universidad de Salamanca

	CUERPO DOCENTE[2]					
Área de Conocimiento	**CU**	**TU**	**CEU**	**TEU**	**PE**	**Total**
Ingeniería de Sistemas y Automática	2	1	0	4	4	***11***
Lenguajes y Sistemas Informáticos	0	3	1	13	9	***26***
Ciencias de la Computación e Inteligencia Artificial	1	0	0	2	0	***3***
Arquitectura y Tecnología de Computadores	0	0	0	1	0	***1***
Total	***3***	***4***	***1***	***20***	***13***	***41***

Tabla 2.2. Profesorado del Departamento de Informática y Automática (en marzo de 2000)

En la Tabla 2.3 se muestran los Centros donde el Departamento tiene docencia asignada.

[2] **CU** = Catedráticos de Universidad, **TU** = Titulares de Universidad, **CEU** = Catedráticos de Escuela Universitaria, **PE** = Profesorado Extraordinario (Ayudantes + Asociados).

Centro	Titulación
Facultad de Ciencias	**Ingeniería Técnica en Informática de Sistemas**
	Ingeniero en Informática (2º Ciclo)
	Licenciatura en Físicas
	Licenciatura en Matemáticas
	Licenciatura en Geología
	Diplomatura en Estadística
Facultad de Químicas	**Ingeniería Química**
Facultad de Traducción y Documentación	**Diplomatura en Biblioteconomía y Documentación**
	Licenciatura en Documentación (2º ciclo)
	Licenciatura en Traducción e Interpretación
Facultad de Derecho	**Diplomatura en Gestión y Administración de Empresas Públicas**
Facultad de Ciencias Agrarias y Ambientales	**Licenciatura en Ciencias Ambientales**
Escuela Técnica Superior de Ingenieros Industriales (Béjar)	**Ingeniería Técnica Industrial en Mecánica**
	Ingeniería Técnica Industrial en Electrónica
	Ingeniería Técnica Industrial en Electricidad
	Ingeniero Industrial (2º Ciclo)
Escuela UniversitariaPolitécnica (Zamora)	**Ingeniería Técnica Industrial (Especialidad Mecánica)**
	Ingeniería Técnica en Obras Públicas (Especialidad Construcciones Civiles)
	Ingeniería Técnica Agrícola (Especialidad Industrias Agrarias y Alimentarias)
	Arquitectura Técnica

Tabla 2.3. Titulaciones en las el Departamento de Informática y Automática imparte docencia

Se puede ver, por tanto, a la década de los 90 como una época de gran dinamismo y crecimiento en la que, sin abandonar las tareas de investigación, los esfuerzos se han centrado en la puesta en marcha de las nuevas titulaciones, con un marcado carácter aplicado. Esto supone, no solamente un gran esfuerzo de trabajo, sino también de planteamiento en tanto que se produce una transición desde un entorno con un elevado carácter teórico hacia uno nuevo con un enfoque mucho más aplicado.

Así, con la creación de nuevas titulaciones, se ha producido un importante cambio y crecimiento de la actividad docente. Además, se puede señalar que tanto en las titulaciones existentes hasta ese momento (Ciencias Físicas, Ingenierías Técnicas,...),

como en las de nueva creación, ha existido una importante dinámica de modificación de planes de estudio que ha hecho de esta década más activa desde un punto de vista docente.

La actividad docente del Departamento se completa con el Programa de Doctorado que en él se imparte. Esta tarea es un punto de conexión de gran importancia entre los dos objetivos principales de la Universidad: *docencia e investigación*. Con la creación del Departamento de Informática y Automática se pone en marcha el Programa de Doctorado con el mismo nombre. En la actualidad están vigentes los Programas correspondientes a los bienios *1998-2000* y *1999-2001*. Los cursos de ambos Programas aparecen en la Tabla 2.4 y en la Tabla 2.5 respectivamente.

El Programa de Doctorado de Informática y Automática está orientado a la formación con profundidad en aspectos, tanto teóricos como prácticos, relativos a las materias englobadas en las áreas de conocimiento que pertenecen al Departamento. Los distintos cursos engloban en sus programas temas de actualidad, y en lo posible relacionados con el trabajo desarrollado por los grupos de investigación.

Titulo del Curso	Créditos
Diseño de Sistemas Borrosos	2
Extensiones de la Lógica de Primer Orden	3
Bases de Datos Distribuidas	2
Planificación y Gestión de Proyectos	2
Reconocimiento Automático de Habla	2
Entornos de Diseño Orientados a Objetos	2
Herramientas Matemáticas para Doctorandos	1
Modelado y Regulación de Procesos	2
Estabilidad de Sistemas Lineales y no Lineales	2
Sistemas en Tiempo Real	2
Técnicas de Inteligencia Artificial Aplicadas al Control	1
Optimización del Diseño, Operación y Control de Procesos Industriales.	1
Técnicas de Control Avanzado	2
Modelos Avanzados de Recuperación de la Información	4
Computación Evolutiva	4
Metodología de la Investigación en Informática	1
Minería de Datos	2
Sistemas Orientados a Objetos	2
Técnicas de Programación Paralela	2
Control e Instrumentación de Instalaciones Industriales	2
Control Óptimo y Filtrado	2
Robótica Industrial	2
Metodología de Diseño de Sistemas de Control Asistido por Ordenador (CACSD)	2

Tabla 2.4. Cursos del programa de doctorado de Informática y Automática (Bienio 1998-2000)

Titulo del Curso	Créditos
Computación Evolutiva	4
Redes Neuronales	4
Extensiones de la Lógica de Primer Orden	3
Reconocimiento Automático de Habla	3
Entornos de Diseño Orientados a Objetos	3
Optimización de Procesos	4
Sistemas Avanzados de Producción	4
Modelos Avanzados de Recuperación de la Información	4
Metodología de la Investigación en Informática	1
Minería de Datos	3
Control de Procesos Clásico y Avanzado	4
Control Óptimo y Filtrado	2

Tabla 2.5. Cursos del programa de doctorado de Informática y Automática (Bienio 1999-2001)

2.1.4 El alumnado

A la hora de hacer el análisis del entorno educativo no puede olvidarse el elemento clave en la actividad universitaria: ***los alumnos***. A ellos va dirigida la labor del docente, y son ellos quienes han de cubrir los objetivos y realizar las actividades oportunas. Por tanto, es importante tener un conocimiento del alumnado, y para ello es necesario hacer un estudio previo de sus características y sus motivaciones.

Desde que comenzó a impartirse la **Diplomatura de Informática** en la Universidad de Salamanca, curso 1989/1990, el número de solicitudes de matrícula ha sido muy elevado, tal y como queda reflejado en los datos de la Tabla 2.6. Sin embargo, la Universidad no podía atender esta fuerte demanda si quería ofrecer unos mínimos de calidad en su enseñanza. Por esta razón el número de admitidos se limitó inicialmente a 100, y esta es la cifra que, aproximadamente, se ha venido manteniendo en los años siguientes.

	89/90	90/91	91/92	92/93	93/94	94/95	95/96	96/97	97/98	98/99	99/2000
Solicitudes	*476*	*297*	*356*	*517*	*576*	*531*	*573*	*548*	*470*	*428*	*339*

Tabla 2.6. Número de solicitudes de matrícula en las que se eligen, en primera opción, los estudios de Informática (1er ciclo)[3]

Inicialmente, los estudios de Informática estaban restringidos a alumnos que hubieran cursado **C.O.U.** En el curso 1992/1993 se amplió también a estudiantes procedentes de la **Formación Profesional**. La proporción de plazas asignadas

[3] Los datos de las tablas han sido facilitados por el Centro de Proceso de datos de la Universidad.

actualmente viene en la actualidad establecida por las disposiciones legales vigentes, que las distribuyen del siguiente modo:

- 60% de alumnos con selectividad superada.
  - Tienen preferencia los alumnos procedentes de la opción Científico-Técnica, Opción A para los alumnos procedentes de COU y opción 1 para los de LOGSE.
- 30% de alumnos procedentes de Formación Profesional.
  - Procedentes de las ramas *Administrativa y Comercial* y *Electricidad y Electrónica* de FP II.
  - Procedentes de diversos módulos profesionales de nivel II como: *Actividades Socioculturales*, *Administración Empresarial*, *Equipos Informáticos*, *Robótica y Automática*...
  - De ciclos formativos de grado superior, como *Desarrollo de Proyectos Mecánicos*, *Sistemas de Telecomunicación e Informáticos*, *Mantenimiento de Equipo Industrial*, *Diseño Producción Editorial*...
- 5% de titulados.
- 5% de extranjeros con selectividad superada en España.

En función de esta distribución, de la demanda y de la limitación anual de matriculados antes mencionada, se establecen cada año unas calificaciones mínimas para poder cursar esta titulación. Estas calificaciones son las que se muestra en la Tabla 2.7.

	92/93	93/94	94/95	95/96	96/97	97/98	98/99	99/2000
Grupo General	*5,93*	*6,14*	*5,60*	*6,04*	*6,17*	*6,46*	*6,60*	*6,79*
Formación Prof.	*6,00*	*6,81*	*6,54*	*6,70*	*6,24*	*5,90*	*6,25*	*6,10*
Otras titulaciones	*1,58*	*1,72*	*1,90*	*1,12*	*1,79*	*1,79*	*1,79*	*1,52*

Tabla 2.7. Calificaciones mínimas necesarias para cursar los estudios de Informática de 1er ciclo

La titulación de Ingeniero en Informática (2º ciclo) comenzó a impartirse en el curso académico 1998/1999, siendo el número de solicitudes, como en el caso de los estudios de 1er ciclo, muy superior al número de admitidos, que en este caso es únicamente de 40 alumnos por curso. Los datos sobre solicitudes de matrícula y puntuaciones mínimas de acceso se recogen en la Tabla 2.8.

	98/99	99/2000
Número de solicitudes	*82*	*88*
Puntuación mínima	*1,49*	*1,44*

Tabla 2.8. Solicitudes y puntuaciones mínimas relativas a los estudios de Informática de 2º ciclo

Para acceder al 2º ciclo se debe estar en posesión del título de Ingeniero Técnico en Informática de Sistemas o de Gestión por la Universidad de Salamanca u otras Universidades donde no exista 2º ciclo. Los alumnos procedentes de la Universidad Pontificia de Salamanca deben tener un contrato formalizado de trabajo con una duración no inferior a un año en el momento de la matricula.

Teniendo en cuenta todos estos datos podemos llegar a las siguientes conclusiones:

- El interés por realizar este tipo de estudios es muy grande, hay una demanda mucho mayor que el número máximo de admitidos, por lo que sólo consigue cursarlos una parte de los alumnos que los han elegido como primera opción. Esto presupone, en principio, que existe una buena motivación por su parte; sin embargo, en el caso de la titulación media, también hay que tener en cuenta que es una carrera que resulta atractiva por otras razones, como por ejemplo por el hecho de que sean estudios de ciclo corto, así como porque la posibilidad de incorporación al mundo laboral de este tipo de titulados es, en términos generales, mayor que la de los procedentes de una gran parte de las disciplinas universitarias.
- Las calificaciones mínimas de acceso a la titulación de primer ciclo muestran una clara tendencia creciente. De ello puede deducirse que la preparación y la capacidad de los estudiantes son, en principio, cada vez mejor. Las puntuaciones requeridas para acceder a la titulación de Ingeniero en Informática revelan que estos alumnos tienen una sólida base para realizar los estudios superiores.
- En la Ingeniería Técnica, la diversa procedencia de los alumnos hace que los grupos no sean homogéneos en cuanto a su formación inicial. En general, los alumnos procedentes de C.O.U. tienen una mejor base teórica en materias tales como matemáticas o física, pero menos específica en informática. Aunque cada vez es menos frecuente, en ocasiones sus conocimientos iniciales en informática son prácticamente nulos. En cambio los alumnos de F.P. llegan con una mayor preparación práctica y un conocimiento en las asignaturas de contenido informático, si bien presentan una falta de base matemática. Su mentalidad va dirigida a la práctica profesional específica y, en general, no muestran interés en los fundamentos teóricos. En cuanto a los alumnos procedentes de la LOGSE, al no ser todavía su implantación total, resulta prematuro evaluar sus efectos en la formación. Los titulados y extranjeros, suelen tener conocimientos en informática y bastante interés en su aprendizaje, pero son dos grupos muy poco significativos dentro del total de alumnos del curso. Esta falta de homogeneidad supone una dificultad para el docente, y en particular en las asignaturas de contenido fuertemente teórico.

- En el segundo ciclo, aunque la mayoría de los alumnos provienen de la Universidad de Salamanca, hay un pequeño grupo de alumnos procedentes de otras Universidades, en las que los planes de estudio son diferentes, por lo que será necesario hacer un esfuerzo para que estos alumnos puedan seguir aquellas asignaturas en las que no tienen base suficiente.
- A diferencia de lo que ocurre en la titulación superior, en los tres cursos de la Ingeniería Técnica el número de alumnos por clase es bastante elevado; esto dificulta la labor docente, ya que impide la utilización de ciertos métodos o recursos didácticos. Condiciona de forma importante la participación activa durante las clases y también la realización de prácticas.

2.1.5 La infraestructura

Los medios materiales de apoyo a la docencia y a la investigación con que cuentan los estudios de Informática se encuentran en un proceso de continua actualización, tanto en calidad de los equipos como en su cantidad. Esta renovación es absolutamente necesaria debido a la rápida evolución que sufren los sistemas informáticos, que en un período de unos pocos años pasan a estar completamente obsoletos.

Para impartir las clases teóricas y de problemas las aulas cuentan con proyectores de transparencias y de diapositivas, a parte, cómo no, de la clásica pizarra.

Para la realización de prácticas en la ***Ingeniería Técnica en Informática de Sistemas*** se dispone en la actualidad de cinco aulas de ordenadores con un total de unos 140 computadores. Cuatro de estas aulas poseen computadores compatibles IBM y los de la otra son Macintosh. Todos ellos se encuentran conectados entre sí mediante una red de área local de tipo Ethernet 10BaseT, a través de la cual es posible conectarse a diferentes estaciones de trabajo entre las que destacan:

- Una estación **HP 9000/821 D250**, con 288 MB de RAM y 15 GB de disco duro. El software más relevante instalado en esta máquina es:
  - *Sistema Operativo HP-UX B.11.00*
  - *Compiladores de C y Fortran*
  - *Sistema Gestor de Base de datos INGRES 2.0*
- Una estación **HP 715/33**, con 80 MB de RAM y 2 GB de disco duro.
- Una estación **HP Apollo Serie 400**, con 16 MB de RAM y 300 MB de disco duro.

Las aulas de informática, gestionadas por dos técnicos especialistas, están a disposición de los alumnos que cursan la titulación, tanto durante el horario de prácticas de las asignaturas como también en el tiempo asignado como de libre utilización. Pueden trabajar en los diferentes entornos y hacer uso los diversos lenguajes de programación y las aplicaciones disponibles. Estas aulas cuentan además con un cañón de proyecciones que facilite enormemente la tarea del profesor en las prácticas guiadas.

Para los alumnos que realizan su ***proyecto fin de carrera*** se dispone además del denominado **Laboratorio de Proyectos**, que cuenta con cuatro estaciones Silicon Graphics MIPS R4600 con 96 MB de RAM, 2,5 GB de disco, S.O. IRIX 6.2, cinco Pentium, un Macintosh 7600 PowerPC 604, un iMAC, escáner e impresoras.

Se dispone también de un *laboratorio de Robótica* dedicado principalmente a labores de investigación y proyectos fin de carrera, dotado con los siguientes elementos:

- Una estación **SUN ULTRA SPARC II Creator**, con 128 MB de RAM y 4 GB de disco, destacando el siguientes software instalado:
  - S O Solaris 2.7
  - Matlab 5.3 Release 11
  - ACSL 11.4.1
- Una estación **SUN SPARC Station 5**, con 64 MB de RAM y 7GB de disco, S O Solaris 2.5.
- 3 Pentium (>100 MHz), Sistemas Operativos Windows, Linux, SCO, RT-Match (Tiempo Real).
- 1 Célula CIM (*Computer Integrated Manufacturing*) formada por:
  - 2 Robots tipo PUMA
  - Cinta transportadora (Bosch)
  - Controladores industriales (Allen Bradley)
  - Autómata programable (SLC-500)
  - Control de movimientos de la cinta (IMC-110)
  - Módulos de entradas y salidas
  - Sistema de visión
    - Cámara (SONY)
    - Tarjeta de procesamiento de imagen (Matrox IMAGE-LC)
    - Procesador digital de señal (DicomLab)

Por otra parte, los alumnos de la ***Ingeniería Informática*** disponen de un laboratorio para su uso exclusivo, con el siguiente equipamiento:

- Servidor UNIX **Silicon Origin200**, Tetraprocesador MIPS RISC R10000 de 64 bits
  - Memoria RAM de 769 MB
  - Disco interno de 18 GB Ultra Fast Wide SCSI (9 + 9 GB)
  - 2 tarjetas Ethernet 10BASET/100BASETX
  - Interfaz Ultra Fast/Wide SCSI-2 a 40 MB/s
  - 3 slots PCI y 6 slots XIO de expansión
  - Destacando el siguiente software instalado:
    - Sistema Operativo IRIX 6.5
    - Software de desarrollo de Silicon Graphics Varsity versión Nodelock
    - Sistema Gestor de Base de Datos ORACLE 8
    - Herramientas de diseño y desarrollo de aplicaciones: ORACLE DEVELOPER/2000 y ORACLE DESIGNER/2000

- Servidor NT Pentium II de Fujitsu
  - Doble procesador Pentium II a 400 MHz
  - 512 Kb de caché por procesador
  - Disco duro Ultra Wide SCSI de 9 Gb
  - 128 MB de memoria SDRAM
  - Adaptador gráfico SVGA AGP con 2 MB de memoria
  - 2 adaptadores de red Ethernet Intel Etherexpress 100BASETX/10BASET PCI
  - Sistema Operativo Windows NT Server, version 4.0
- 25 PC´s Pentium II a 266 MHz
  - 64 MB de memoria DIMM SDRAM
  - Disco duro de 3,2 GB Ultra DMA
  - Tarjeta de red 3COM Fast Ethernet 10/100
  - Sistemas Operativos Windows NT WorkStation y Linux
- Red Fast Ethernet 100BASETX con los siguiente elementos de red:
  - Armario de cableado estructurado
  - 1 Hub 100BASETX de 3COM (Office Connect 4 TP400 100 BTX NG)
  - 1 Switch 3300 de 3COM 24 port 10/100
  - 1 Router 3COM SuperStack II Netbuilder 432

Además, de los medios informáticos, la utilización de las bibliotecas debería ser práctica habitual de los alumnos, puesto que la consulta de libros y publicaciones es un complemento fundamental a la enseñanza recibida en las clases. En concreto, los estudiantes pueden hacer uso de las diferentes bibliotecas de la Universidad de Salamanca, y en particular de la propia de la Facultad de Ciencias, que es la que dispone de la mayor parte de los textos relacionados con las materias que se imparten en las titulaciones de Informática. En un futuro próximo esta biblioteca dispondrá de unas nuevas instalaciones, con lo que se ampliará, se dotará de nuevos fondos bibliográficos y también de terminales para su consulta.

2.2 El marco curricular

2.2.1 Perspectiva histórica de los estudios de Informática

La investigación en máquinas capaces de calcular de manera automática ha tenido gran actividad desde antes del siglo XX, pero es en la década de los 40, principalmente como respuesta a las necesidades militares, cuando surgieron las primeras computadoras electrónicas. Aunque fueron desarrolladas en entornos universitarios, hasta mediados de la década de los 50 no aparecen los primeros centros de computación universitarios. Generalmente estos centros estaban vinculados a Departamentos de Matemáticas o Ingeniería, y servían de soporte a la investigación en los mismos.

Los primeros programas académicos en Instituciones Superiores de educación aparecieron a mediados de la década de los 50. Su principal orientación era la formación de los usuarios de computadoras y por ello su contenido consistía en temas relacionados con el manejo de los equipos. Entre estos centros cabe destacar las siguientes Universidades: *University of Michigan*, *University of Houston*, *Stanford University* y *Purdue University*. En Europa la educación en Informática se desarrolló más o menos de forma simultánea a la de EE.UU. Algunas de las primeras computadoras fueron instaladas, e incluso construidas, en las propias Universidades, con el fin de servir a los propósitos de investigación de los diferentes Departamentos. A partir de 1965 comienzan a aparecer titulaciones de Informática en Gran Bretaña, Francia y Alemania.

La primera computadora instalada en España fue una IBM 650 (*modelo del año 1953*) en la Compañía Nacional Telefónica, en el año 1958. En 1962 empezaron a introducirse las primeras computadoras en empresas privadas. En esta época, la enseñanza del manejo y fundamentos de estas máquinas corrió a cargo de las propias empresas constructoras.

En el año 1969 comienza la enseñanza de Informática de manera oficial en España con la creación del **Instituto de Informática** en Madrid. En él se imparte una carrera de cinco años, de los cuales los tres primeros son comunes y los dos últimos de especialización. En 1971 se crea una delegación de este Instituto en San Sebastián, y en 1972 se forma en la Universidad Autónoma de Barcelona un Departamento de Informática. De forma paralela, se incluyen en diversas titulaciones universitarias de carácter técnico o científico asignaturas específicas de Informática. A finales de los años setenta se empiezan a impartir Estudios Superiores de Informática en la Universidad. Éstos fueron concebidos como diplomaturas y licenciaturas. Actualmente, según las directrices del Ministerio de Educación y Ciencia sobre nuevas titulaciones, se han reconvertido las titulaciones a Ingenierías Técnicas y Superiores.

En el ámbito internacional, en la década de los 60, algunas universidades norteamericanas (*Harvard, MIT, Columbia, Pensylvania*) incorporaron currículos completos de Informática dentro de sus enseñanzas. En ellos se plantearon dos enfoques, por un lado el que se denominó "*Computer Science*", más orientado a la programación y a la algoritmia, y por otro lado el enfoque conocido como "*Computer Engineering*" enfocado hacia la arquitectura y tecnología de computadores.

Desde estas fechas hasta nuestros días se han realizado numerosos esfuerzos por redactar un documento que unifique los diferentes currículos en Informática. Este documento debería ser lo suficientemente flexible para poder adaptarlo a situaciones particulares de disponibilidad de medios materiales, entorno económico-social, plantilla de profesorado... Existen dos sociedades destacadas en éste esfuerzo: **ACM** (*Association of Computer Machinery*) e **IEEE** (*Institute of Electrical and Electronic Engineers*). Como resultado de esta iniciativa surge el *Computing Curricula 1991*,

ACM/IEEE-CS'91 [Tucker et al., 1990], referencia obligada en la creación de Planes de Estudio de Informática[4].

Del repaso de los diferentes currículos de ámbito internacional que se han generado a lo largo de estos años, se desprende, que lo que en España se engloba bajo el título de Informática se corresponde con diversas disciplinas en el mundo anglosajón. Las más extendidas y aceptadas actualmente son: *Computer Science, Computer Engineering, Information Systems* y *Software Engineering*. Cada una de estas titulaciones puede tener diferentes enfoques en distintas Universidades.

Los conceptos que se ocultan detrás de estos nombres no tienen una significación común para todo el mundo, siendo su interpretación más adecuada la siguiente [Camps, 1999]:

- **CS – Computer Sicence – Ciencia de la Informática:** Se trata de la Informática como disciplina científica construida sobre fundamentos lógicos y matemáticos. El científico CS es un investigador que desarrolla principios fundamentales, teorías y herramientas formales. Con frecuencia se usa el término CS en un sentido más amplio, casi sinónimo a nuestro término "Informática". Pero la palabra *Science* no ofrece mucha diversificación. Así, se está adoptando el término *Computing* como vocablo global polivalente, más cercano a la palabra Informática.
- **CE – Computer Engineering – Ingeniería de los computadores:** Ingeniería relativa al diseño y construcción de herramientas informáticas, tanto hardware como software. Aplica los principios fundamentales de la Ciencia de la Informática.
- **SE – Software Engineering – Ingeniería del Software:** Actualmente el término CE se tiende a restringir a la ingeniería del hardware, utilizando el término SE para la ingeniería del software.
- **SI – Information Systems – Sistemas de Información:** Se trata de la aplicación de la tecnología informática (*por ejemplo herramientas software desarrolladas por ingenieros del software*) a la gestión de la información en el mundo de las organizaciones (*empresas, instituciones*). Abarca un campo que va desde la planificación estratégica de las tecnologías de la información en la empresa, hasta el diseño/programación de aplicaciones de gestión, es decir comparte terreno con la Ingeniería del Software y la Administración de Empresas.

Las propuestas del ACM/IEEE-CS'91 se centran en recomendaciones para planes de estudios de Informática (*Computing*) que incluyen planes para Ciencia de la

[4] De hecho, los actuales Planes de Estudio de Informática de la mayoría de las Universidades españolas se automanifiestan inspirados en las recomendaciones recogidas en el ACM/IEEE-CS'91 [Camps, 1999].

Informática, Ingeniería de Ordenadores, Ciencia de la Informática e Ingeniería del Software o similares, quedando fuera los Sistemas de Información.

Pero hay sectores que abogan por la separación de la parte de Ciencia de la Informática de la parte de Ingeniería Informática (*Ingeniería del Software*), argumentando que ambas tienen orientaciones distintas por mucho que se influyan y fertilicen mutuamente [Lutz and Naveda, 1997], llegando a cuestionar incluso la validez de cara al futuro de la propuesta curricular ACM/IEEE-CS'91 como núcleo común entre la Ciencia de la Informática y la Ingeniería Informática, abogando por propuestas curriculares independientes [Bagert, 1999].

Quizás uno de los mayores defensores de esta división sea **David Lorge Parnas** que expresa sus dudas de que la formación que se está ofreciendo a los informáticos sea la adecuada para ejercer su labor profesional, necesitándose una correcta definición de la profesión, un reconocimiento de la Ingeniería del Software como una nueva rama de la Ingeniería, una identificación del cuerpo de conocimiento propio de la Ingeniería del Software y una comunicación más efectiva entre las partes en conflicto, de manera que se solucionen las carencias de los ingenieros del software a la hora de aplicar los formalismos oportunos en su labor profesional y se palie el desconocimiento de los científicos sobre lo que significa ser ingeniero [Parnas, 1997], [Parnas, 1999].

El sentir de David L. Parnas puede resumirse en la siguiente cita:

> **"La Ingeniería consiste en el uso de la Ciencia y la Tecnología para construir productos que serán utilizados por otras personas. El software es uno de esos productos. Los ingenieros encuentran sus problemas en la práctica. Los científicos en la literatura. Un 'científico' ve una Máquina de Estados Finita como un modelo de computación, mientras que un 'ingeniero' ve en ella una herramienta de diseño. La mayoría de los científicos que estudian la 'ciencia de la programación' no entienden lo que es un ingeniero. Porque la Ingeniería del Software se basa en la Ciencia de la Informática, creen que la Ingeniería del Software es Ingeniería de los Ordenadores. Esto sería como confundir la Ingeniería Electrónica con la Física. Actualmente la mayoría de titulaciones de Ciencias de la Informática no son titulaciones de ciencias puras, pero tampoco pueden acreditarse como ingenierías. Son soluciones de compromiso y no hacen ninguna de las dos cosas correctamente".**
>
> **David Lorge Parnas. "***Software Engineering: An Unconsummated Marriage***".** Tutorial en el **ESEC'97.** Agosto, 1997.

Pero como era de imaginar no todo el mundo está de acuerdo con esta separación tan drástica de la Ciencia de la Informática y la Ingeniería del Software, buscando una definición de la profesión de informático (*computing*) como una nueva ingeniería única,

no separando las áreas de conocimiento, basada en una nueva propuesta curricular que tome como punto de partida el ACM/IEEE-CS'91, y que unifique un marco de acreditación para los titulados (*ingenieros*) [El-Kadi, 1999].

Como resumen de la postura no separatista puede citarse a **Peter J. Denning**, quizás una de las personas que más influyeron en el *Computing Curricula 91*, que expone en [Denning, 1998]:

> **"La separación entre la teoría y la ingeniería ha sucedido en otras disciplinas porque éstas habían madurado lo suficiente para que hubiera una comunicación fluida entre sus ramas científica, ingenieril y de aplicación. Una separación similar sería un desastre en la Ciencia de la Informática. La segregación de los ingenieros del software provocaría que la comunicación entre ingenieros, teóricos y especialistas en aplicaciones se acabase. La comunicación, no el divorcio, es la respuesta".**

2.2.2 Las titulaciones de Informática: Ley de reforma universitaria

Las enseñanzas universitarias en nuestro país están pasando en la actualidad por una fase de renovación, que se inició en 1983 con la Ley Orgánica 11/1983 de 5 de agosto, de Reforma Universitaria (LRU). Los dos postulados básicos en el proceso de reforma han sido:

- La vertebración de las enseñanzas universitarias en una estructura cíclica que permita la obtención de sucesivos títulos oficiales consecutivos.
- La redefinición de los contenidos formativos y las exigencias académicas de los planes de estudios, intentando con ello acercar la formación universitaria a la realidad social y profesional de nuestro entorno.

Las directrices generales comunes de los planes de estudios se presentan en el Real Decreto 1497/1987 de 27 de Noviembre, publicado en el BOE de 14 de Diciembre. Con ellas se busca una mayor flexibilidad de sus fórmulas y soluciones académicas que permita una mayor rentabilidad de la oferta universitaria, un mejor aprovechamiento del discente y un más amplio abanico de opciones. Para conseguir este fin se plantea:

- La racionalización de la duración de las carreras y de la carga lectiva, hasta el momento excesiva.
- La convicción de que la enseñanza práctica debe asumir una mayor relevancia en la Universidad.
- La incorporación de un sistema de cómputo mediante créditos, que potencia una mayor flexibilidad en el currículo del estudiante.

Resultan así unos estudios caracterizados por ser:

- ***Cíclicos***, es decir, estructurados, como máximo, en tres ciclos. La superación del primero de ellos dará derecho, en cada caso, a la obtención del título de Diplomado, Ingeniero Técnico o Arquitecto Técnico. El segundo corresponderá al título de Licenciado, Ingeniero o Arquitecto, y el tercero al de Doctor.
- ***Modulares***, constituidos por una serie de materias, cada una de las cuales tendrá un valor medido en créditos. Los créditos tienen una correspondencia en horas de docencia, que en la actualidad es de *10* horas por crédito. Para la obtención del título correspondiente a un ciclo, el alumno deberá lograr una determinada cantidad de créditos previamente establecida.
- ***Flexibles***, para ofrecer al alumno la posibilidad de decidir y elaborar su perfil académico, seleccionando entre una serie de materias optativas las que considere más convenientes. Para hacer posible esta flexibilidad y al tiempo asegurar una base común en todos los titulados, se establecen tres tipos de materias:
    - ***Materias Troncales***: Constituyen los contenidos mínimos exigibles a un mismo título oficial, y son por tanto obligatorias en todo el territorio nacional. Deben constituir no menos del 30% de la carga docente total durante el primer ciclo, y al menos el 25% de la misma durante el segundo ciclo. Las Universidades, al establecer los correspondientes planes de estudios, podrán organizar las materias troncales en disciplinas o asignaturas concretas.
    - ***Materias determinadas por la Universidad en sus planes de estudios***: Son materias definidas particularmente por cada Universidad para cada titulación; se dividen a su vez en obligatorias y optativas.
        - ***Materias obligatorias:*** Libremente establecidas por cada Universidad, que las incluye dentro del correspondiente plan de estudios como obligatorias para el alumno.
        - ***Materias optativas:*** Libremente establecidas por cada Universidad, que las incluye en el correspondiente plan de estudios para que el alumno escoja entre las mismas.
    - ***Libre configuración***: Al menos el 10% de la carga lectiva de un plan de estudios deberá quedar abierta para que el estudiante pueda cursar aquellas materias que libremente escoja entre las ofrecidas por la Universidad. La finalidad de esta categoría es potenciar la

formación interdisciplinaria, y se orienta principalmente a materias de carácter general.

El Consejo de Universidades evaluó el cumplimiento por las Universidades de las directrices generales de los planes de estudio, y el contenido científico, técnico o artístico, y la adecuación profesional de los mismos. Esta evaluación ha puesto de manifiesto algunos problemas interpretativos de la normativa establecida por el Real Decreto de 1987. Entre los problemas interpretativos y desajustes se pueden citar los derivados:

- De la distinta duración de los segundos ciclos de estudios que llevan a la misma titulación, que puede llevar a la distorsión del currículo académico del alumno.
- Del incremento excesivo de la troncalidad que puede falsear el contenido homogéneo de las enseñanzas.
- De la tendencia a una especialización excesivamente temprana.
- De la falta de inclusión en los planes de estudio, de materias obligatorias u optativas de carácter complementario o instrumental no específicas de la titulación, pero coherente con la formación básica y general que se exige para el primer ciclo.

Como consecuencia de todo esto se estimó necesario efectuar ciertas aclaraciones e introducir modificaciones parciales en algunos de los artículos del Real Decreto 1497/1987, de 27 de noviembre por medio de la publicación del Real Decreto 1267/1994 de 10 de junio.

Además, estas directrices comunes han sido modificadas recientemente por el Real Decreto 779/1998 de 30 de abril de 1998 (*B.O.E. número 104 de 1 de mayo de 1998*). En él se modifica el concepto de crédito (puede dedicarse hasta un 30% a actividades académicas dirigidas) y se establece en 6 el límite máximo de materias a cursar por los alumnos de forma simultánea.

Una vez establecidas las directrices generales comunes, aplicables a todos los planes de estudio, se establecen las directrices generales propias de Informática. Esto tiene lugar en los Reales Decretos 1459/1990, 1460/1990 y 1461/1990 (B.O.E. número 278 de 20 de noviembre) en los que se contemplan los títulos de Ingeniero en Informática, Ingeniero Técnico en Informática de Gestión e Ingeniero Técnico en Informática de Sistemas (Tablas 2.9, 2.10 y 2.11 respectivamente).

Materias troncales	Créditos	Áreas de conocimiento
PRIMER CICLO		
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados.	6	Ciencias de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de Datos y de la Información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: Ficheros, bases de datos.	12	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Estructura y Tecnología de Computadores. Unidades funcionales: Memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos.	15	Arquitectura y tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica.
Fundamentos Físicos de la Informática. Electromagnetismo, Estado Sólido. Circuitos.	6	Electrónica, Electromagnetismo, Física Aplicada, Física de la Materia Condensada, Ingeniería Eléctrica y Tecnología Electrónica.
Fundamentos Matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos.	18	Álgebra, Análisis Matemático, Ciencias de la Computación e Inteligencia Artificial, Matemática Aplicada
Metodología y Tecnología de la Programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: Descomposición modular y documentación. Técnicas de verificación y pruebas de programas.	15	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Sistemas Operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros.	6	Arquitectura y Tecnología de Computa-dores, Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Teoría de Autómatas y Lenguajes Formales. Máquinas secuenciales y autómatas finitos. Máquinas de Turing. Funciones recursivas. Gramáticas y lenguajes formales. Redes neuronales.	9	Álgebra, Ciencias de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Lenguajes y Sistemas Informáticos y Matemática Aplicada.
SEGUNDO CICLO		
Arquitectura e Ingeniería de Computadores. Arquitecturas paralelas. Arquitecturas orientadas a aplicaciones y lenguajes.	9	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica.
Ingeniería de Software. Análisis y definición de requisitos. Diseño, propiedades y mantenimiento del software. Gestión de configuraciones. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones.	18	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Inteligencia Artificial e Ingeniería del Conocimiento. Heurística. Sistemas basados en el conocimiento. Aprendizaje. Percepción.	9	Ciencias de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática y Lenguajes y Sistemas Informáticos.
Procesadores de lenguaje. Compiladores. Traductores e intérpretes. Fases de compilación. Optimización de código. Macroprocesadores.	9	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Redes. Arquitectura de redes. Comunicaciones.	9	Arquitectura y Tecnología de Computadores, Ciencias de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Ingeniería Telemática y Lenguajes y Sistemas Informáticos.
Sistemas Informáticos. Metodología de análisis. Configuración, diseño, gestión y evaluación de sistemas informáticos. Tecnologías avanzadas de sistemas de información, bases de datos y sistemas operativos. Proyectos de sistemas informáticos	15	Arquitectura y Tecnología de Computadores, Ciencias de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa, Ingeniería de Sistemas y Automática, Ingeniería Telemática, Lenguajes y Sistemas Informáticos y Organización de Empresas.

Tabla 2.9. Título de Ingeniero en Informática

Materias troncales	Créditos	Áreas de conocimiento
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados.	9	Ciencias de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de datos y de la información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: Ficheros, bases de datos.	12	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Estructura y tecnología de computadores. Unidades funcionales: Memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos.	9	Arquitectura y tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica.
Fundamentos matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos.	18	Álgebra, Análisis Matemático, Ciencias de la Computación e Inteligencia Artificial, Matemática Aplicada
Ingeniería de Software de gestión. Diseño, propiedades y mantenimiento del software de gestión. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones de gestión.	12	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Metodología y tecnología de la programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: Descomposición modular y documentación. Técnicas de verificación y pruebas de programas.	15	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Sistemas operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros.	6	Arquitectura y Tecnología de Computadores, Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Técnicas de organización y gestión empresarial. El sistema económico y la empresa. Técnicas de administración y técnicas contables.	12	Economía Financiera y Contabilidad y Organización de Empresa.

Tabla 2.10. Título de Ingeniero Técnico en Informática de Gestión

Materias troncales	Créditos	Áreas de conocimiento
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados.	9	Ciencias de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de Datos y de la Información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: Ficheros, bases de datos.	12	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Estructura y Tecnología de Computadores. Unidades funcionales: Memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos.	15	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica.
Fundamentos Físicos de la Informática. Electromagnetismo, Estado Sólido. Circuitos.	6	Electrónica, Electromagnetismo, Física Aplicada, Física de la Materia Condensada, Ingeniería Eléctrica y Tecnología Electrónica.
Fundamentos Matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos.	18	Álgebra, Análisis Matemático, Ciencias de la Computación e Inteligencia Artificial, Matemática Aplicada
Metodología y Tecnología de la Programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: Descomposición modular y documentación. Técnicas de verificación y pruebas de programas.	12	Ciencias de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.
Redes. Arquitectura de redes. Comunicaciones.	9	Arquitectura y tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Ingeniería Telemática y Lenguajes y Sistemas Informáticos.
Sistemas Operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros.	6	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos.

Tabla 2.11. Título de Ingeniero Técnico en Informática de Sistemas

2.2.3 Las titulaciones de Informática en la Universidad de Salamanca

2.2.3.1 Estudios de primer ciclo

Como se ha comentado anteriormente, la enseñanza oficial de la Informática en España comenzó en el año 1969, con la creación del Instituto de Informática en Madrid. En 1976 es cuando se crean en nuestro país las primeras Facultades de Informática, en Madrid, Barcelona y San Sebastián. Limitándonos al entorno geográfico de Castilla y León, fue la Escuela Universitaria de Ingeniería Técnica Industrial de Valladolid la que primeramente estableció la Sección de Informática, y comenzó a impartir docencia en el año académico 1985-1986.

La Universidad de Salamanca inició este tipo de estudios en el curso 1989-1990, bajo la responsabilidad de la Facultad de Ciencias. La titulación impartida era la de ***Diplomado en Informática***, tal y como se recoge en el Real Decreto 1025/1989 publicado en el B.O.E. de 10 de Agosto de 1989. Se trataba de estudios de primer ciclo, con una duración de tres años académicos. La carga lectiva total era de 227 créditos, de los cuales 190 correspondían a materias obligatorias, 12 a optativas y el resto a asignaturas de libre elección, tal y como se indica en la Tabla 2.12.

Poco más tarde, en concreto en Octubre del año siguiente, se aprueba el Real Decreto 1561/1990, publicado en el B.O.E. de 20 de Noviembre de 1990, en el que se establece el título oficial de ***Ingeniero Técnico en Informática de Sistemas***. En este decreto se recogen las directrices generales de los planes de estudio que deben cursarse para su obtención y homologación según propuesta del Consejo de Universidades y del Ministerio de Educación y Ciencia.

Siguiendo dichas directrices, la Universidad de Salamanca procede a la implantación de este plan, de acuerdo con la resolución de 23 de Noviembre de 1992, publicada en el B.O.E. de 7 de Enero de 1993. La nueva titulación de Ingeniería Técnica en Informática de Sistemas comienza su andadura en el curso 1993-1994 y está vigente en la actualidad.

El comienzo de estos nuevos estudios supuso, entre otras cosas, una redistribución de la carga lectiva, que pasó a ser de 213 créditos, repartidos tal y como aparece reflejado en la Tabla 2.12. Otra de sus características más destacadas fue la aparición de la asignatura **Proyecto** como materia obligatoria de 12 créditos en tercer curso, tal y como puede verse en la Tabla 2.13. En esta tabla se muestran además todas las asignaturas troncales (T) y obligatorias (Obl), indicando el curso en el que se imparten y el número y distribución de los créditos asociados a cada una de ellas. Las materias optativas (Opt) y de libre configuración (LC) no se detallan, únicamente se indica el número total de créditos de cada tipo necesarios en cada curso.

	Troncal	Obligat.	Optativa	Libre C.	TOTAL
Diplomatura	**190**	**-**	**12**	**25**	**227**
Ingeniería Técnica (Plan 1992)	**159**	**12**	**18**	**24**	**213**
Primer curso	60	-	-	8	68
Segundo curso	57	-	9	8	74
Tercer curso	42	12	9	8	71
Ingeniería Técnica (Plan 1997)	**100.5**	**49.5**	**30**	**21**	**201**
Primer curso	42	24	-		66
Segundo curso	46.5	10.5	12	6	75
Tercer curso	12	15	18	15	60

Tabla 2.12. Comparación de la carga lectiva de la titulación de Informática en los diferentes planes de estudios

	Asignatura	Créditos			
		Tipo	Totales	Teóricos	Prácticos
PRIMER CURSO	Álgebra	T	12	6	6
	Análisis Matemático	T	12	6	6
	Electrónica	T	9	6	3
	Fundamentos Físicos de la Informática	T	9	6	3
	Introducción a los Computadores	T	9	6	3
	Programación I	T	9	6	3
		LC	8		
SEGUNDO CURSO	Estructura y Tecnología de Computadores	T	12	6	6
	Sistemas Operativos	T	12	9	3
	Bases de Datos	T	9	6	3
	Estadística	T	9	6	3
	Programación II	T	9	6	3
	Estructura de la Información	T	6	4.5	1.5
		Opt	9		
		LC	8		
TERCER CURSO	Ampliación de Matemáticas	T	9	6	3
	Redes de Ordenadores	T	9	6	3
	Teoría de Autómatas y Lenguajes Formales	T	9	6	3
	Teoría de la Señal y Transmisión de Datos	T	9	6	3
	Ingeniería del Software	T	6	4.5	1.5
	Proyecto	Obl	12	0	12
		Opt	9		
		LC	8		

Tabla 2.13. Asignaturas del plan de estudios de 1992

	Asignatura	Créditos			
		Tipo	Totales	Teóricos	Prácticos
PRIMER CURSO	Fundamentos Físicos de la Informática	T	7.5	6	1.5
	Álgebra	T	6	3	3
	Cálculo Diferencial	T	6	3	3
	Programación	T	6	6	0
	Sistemas Informáticos	Obl	6	3	3
	Laboratorio de Programación	Obl	4.5	0	4.5
	Electrónica	T	9	5	4
	Algoritmia	T	7.5	4.5	3
	Cálculo Integral	Obl	7.5	4.5	3
	Álgebra Computacional	Obl	6	3	3
SEGUNDO CURSO	Estadística	T	7.5	4.5	3
	Unidades Funcionales del Ordenador	T	7.5	5	2.5
	Matemática Discreta	T	6	3	3
	Sistemas Operativos	T	6	6	0
	Estructuras de Datos	T	6	3	3
	Diseño de Bases de Datos	T	4.5	4.5	0
	Lenguajes Formales	T	4.5	3	1.5
	Sistemas de Bases de Datos	T	4.5	1.5	3
	Transmisión de Datos	Obl	6	4.5	1.5
	Laboratorio de Sistemas Operativos	Obl	4.5	0	4.5
		Opt	12		
		L.C.	6		
TERCER CURSO	Redes	T	7.5	4.5	3
	Informática Teórica	T	4.5	3	1.5
	Ingeniería del Software	***Obl***	***6***	***4.5***	***1.5***
	Proyecto	Obl	9	0	9
		Opt	18		
		L.C.	15		

Tabla 2.14. Asignaturas del plan de estudios de 1997

En el curso 1997 la Universidad de Salamanca se volvió a replantear la estructura y contenidos de varias titulaciones, entre ellas la Ingeniería Técnica en Informática de Sistemas. Dicha revisión se realizó atendiendo a:

- Las recomendaciones del Consejo de Universidades (B.O.E. de 17 de Enero de 1997).
- Las normas que han modificado el Real Decreto 1497/87 hasta el momento presente.
- La evaluación de la Comisión a la que la Junta de la Facultad de Ciencias encargó el análisis del Plan de 1992 una vez implantados los tres años académicos de que consta; esta evaluación ha tenido en cuenta las opiniones

del profesorado y de los alumnos, principalmente sobre el rendimiento académico.

Con ello se elabora un nuevo plan, que fue aprobado por resolución de 15 de octubre de 1997; publicándose en el B.O.E. de 4 de Noviembre de 1997.

Las modificaciones más destacadas fueron: *una estructuración básicamente cuatrimestral* y *una nueva disminución y reorganización de la carga lectiva*.

La distribución global de créditos de este nuevo plan aparece también en la Tabla 2.12. En la Tabla 2.14 se muestran con detalle las materias troncales y obligatorias de que consta y su distribución concreta.

Con el plan de 1997 se amplía la flexibilidad de los estudios, reduciendo la proporción de materias troncales en favor de las obligatorias y optativas. Ello permite una mayor libertad, tanto a la Universidad, a la hora de definir sus programas, como al alumno para elegir su especialización. Destaca, además, que la optatividad aumenta con el curso, para tener en cuenta que el estudiante, a medida que avanza en los estudios, tiene un mayor conocimiento y, por tanto, más capacidad de elección. La Tabla 2.15 muestra la relación concreta de las asignaturas optativas de este plan de estudios.

Asignatura	Créditos		
	Totales	Teóricos	Prácticos
Arquitecturas Avanzadas	6	4.5	1.5
Programación Orientada a Objetos	***6***	***3***	***3***
Interfaces Gráficas	6	3	3
Administración de Sistemas Informáticos	6	3	3
Control de Procesos	6	4.5	1.5
Tecnología de Control	6	3	3
Modelado y Simulación	6	4.5	1.5
Introducción a la Economía de Empresa	6	4.5	1.5
Lógica Matemática	6	4.5	1.5
Modelos Estadísticos Lineales	6	4.5	1.5
Periféricos	6	3	3
Sistemas de Transmisión de Señal	6	4.5	1.5
Paquetes Estadísticos	6	1.5	4.5

Tabla 2.15. Asignaturas optativas del plan de estudios de 1997

Otra de las características que se ha buscado en este plan es disminuir la carga lectiva del sexto cuatrimestre para que los alumnos puedan dedicarse más intensamente a la elaboración del ***proyecto***.

La implantación de este nuevo plan de la Ingeniería Técnica en Informática de Sistemas tuvo lugar en el curso académico 1997-1998.

2.2.3.2 Estudios de segundo ciclo

La titulación de Ingeniería en Informática comienza a impartirse en la Universidad de Salamanca en el curso académico 1998-1999, siendo el Centro Universitario responsable de la organización del plan de estudios la Facultad de Ciencias.

Para acceder a estos estudios de segundo ciclo es necesario estar en posesión del título de Ingeniero Técnico en Informática de gestión o de sistemas, de acuerdo con la orden de 11 de septiembre de 1991 (BOE de 26 de septiembre).

El plan de estudios aprobado por la Universidad de Salamanca fue homologado por acuerdo de 27 de octubre de 1998, de la Comisión Académica del Consejo de Universidades. La resolución de la Universidad de 10 de junio de 1999 aparece publicada en el BOE número 156 de 1 de julio de 1999.

La elaboración del plan de estudios de dicha titulación se basó en las directrices generales de los planes de estudio (Real Decreto 1497/1987 de 27 de noviembre) y en las directrices para la obtención del título oficial de Ingeniero en Informática (Real Decreto 1459/1990, BOE de 20 de noviembre) que se recogen en la Tabla 2.9.

La carga lectiva total es de 127 créditos distribuidos en dos cursos. La repartición de los créditos aparece en la Tabla 2.16.

Curso	Materias troncales	Materias obligatorias	Materias optativas	Créditos de libre configuración	Trabajo fin de carrera	Totales
1º	*36*	*9*	*12*	*6*	*0*	*63*
2º	*33*	*6*	*18*	*7*	*0*	*64*
Total 2º ciclo	**69**	**15**	**30**	**13**	**0**[5]	**127**

Tabla 2.16. Distribución de la carga lectiva global del segundo ciclo

El reparto de las asignaturas en los dos cursos en que se divide el segundo ciclo queda reflejado en la Tabla 2.17.

En dicha tabla puede observarse que la Universidad de Salamanca, además de las asignaturas correspondientes a las materias troncales, ha incluido dos asignaturas obligatorias**, Ampliación de Sistemas Operativos** y **Ampliación de Bases de Datos**, para completar la formación de los estudiantes de esta titulación. Asimismo se oferta un gran número de asignaturas optativas que figuran en la Tabla 2.18.

En segundo curso aparecen dos asignaturas troncales cuya carga crediticia es totalmente práctica, la asignatura **Sistemas de Información** que se cursa en el primer semestre y la asignatura **Proyecto** que se cursa en el segundo. Para examinarse de esta

[5] El trabajo fin de carrera está incluido dentro de la troncalidad (6 créditos).

última será necesario tener aprobados el resto de los créditos, cualquiera que sea la naturaleza de éstos.

La materia troncal ***Ingeniería del Software*** se ha dividido en dos asignaturas, **Análisis de Sistemas**, que se imparte el primer curso y **Administración de Proyectos Informáticos** que aparece en segundo curso.

	Asignatura	Créditos			
		Tipo	Totales	Teóricos	Prácticos
PRIMER CURSO	Arquitectura e Ingeniería de Computadores	T	9	6	3
	Análisis de Sistemas	T	9	6	3
	Procesadores de Lenguaje	T	9	6	3
	Redes	T	9	6	3
	Ampliación de Sistemas Operativos	Obl	9	4,5	4,5
SEGUNDO CURSO	Administración de Proyectos Informáticos	T	9	6	3
	Inteligencia Artificial e Ingeniería del Conocimiento	T	7.5	5	2.5
	Sistemas de Información	T	9	0	9
	Proyecto	T	6	0	6
	Estructuras de Datos	T	6	3	3
	Ampliación de Bases de Datos	Obl	6	4.5	1,5

Tabla 2.17. Asignaturas de la titulación de Ingeniero en Informática (plan de 1999).

Asignatura	Créditos		
	Totales	Teóricos	Prácticos
Programación Paralela y Distribuida	6	3	3
Administración de Sistemas Informáticos	6	2	4
Procesamiento de Imágenes	6	3	3
Informática Gráfica	6	3	3
Microelectrónica	6	4.5	1.5
Diseño de Circuitos Digitales	6	3	3
Técnicas de Investigación Operativa	6	3	3
Reconocimiento de Patrones	6	3	3
Técnicas de Control de Calidad	6	3	3
Criptografía	6	3	3
Lógica Computacional	6	3	3
Cálculo Numérico	6	3	3
Tecnología de Control	6	4.5	1.5
Robótica	6	4.5	1.5
Lógica para la Informática y la Inteligencia Artificial	6	1.5	4.5

Tabla 2.18. Asignaturas optativas de la titulación de Ingeniero en Informática

2.3 Referencias

[Bagert, 1999] Bagert, Donald J. "*Talking the Lead in Licensing Software Engineers*". Communications of the ACM, 42(4): 27-29. April, 1999.

[Camps, 1999] Camps Paré, Rafael. "*¿Qué Informática Se Enseña en la Universidad? (Primera Parte)*". Novática. Nº 141: 48-51. Septiembre/Octubre, 1999.

[Denning, 1998] Denning, Peter J. "*Computer Sciencie and Software Engineering: Filing for Divorce?*". Communications of the ACM, 40(8):128. August, 1998.

[El-Kadi, 1999] El-Kadi, Amr. "*Stop that Divorce*". Communications of the ACM, 42(12):27-28. December, 1999.

[Lutz and Naveda, 1997] Lutz, Michael J. and Naveda, J. Fernando. "*The Road Less Traveled: A Baccalaureate Degree in Software Engineering*". In Proceedings of the twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97). (February 27 - March 1, 1997, San Jose, CA USA). ACM. Pages 287-291. 1997.

[MEC, 1983] Ministerio de Educación y Ciencia. "*Ley Orgánica de Reforma Universitaria*". Servicio de Publicaciones del MEC, 1993.

[Ortega, 1982] Ortega y Gasset, José. "*Misión de la Universidad*". En Paulino Garagorri *Obras de J. Ortega y Gasset*, edición nº 22. El Arquero, Alianza Editorial, 1982.

[Parnas, 1997] Parnas, David Lorge. "*Software Engineering: An Unconsummated Marriage*". Communications of the ACM, 40(9):128. September, 1997.

[Parnas, 1999] Parnas, David Lorge. "*Software Engineering Programs Are Not Computer Science Programs*". IEEE Software, 16(6):19-30. November/December, 1999.

[Russell, 1987] Russell, B. "*La Perspectiva Científica*". 2ª edición. Editorial Ariel, 1987.

[Russell, 1997] Russell, B. "*Respuestas*". Edición de Lee Eisler. Península, 1997.

[Savater, 1998] Savater, F. "*El Valor de Educar*". 9ª edición. Editorial Ariel, S.A., 1998.

[Sierra, 1986] Sierra Bravo, R. "*Tesis Doctorales y Trabajos de Investigación Científica. Metodología General de su Elaboración y su Documentación*". Paraninfo, 1986.

[Tucker et al., 1990] Tucker, Allen B. (Editor and Co-chair), Barnes, Bruce H. (Co-chair), Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke, Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*Computing Curricula 1991. Report of the ACM/IEEE-CS Joint Curriculum Task Force*". ACM. http://www.acm.org/education/curr91/homepage.html. December, 1990.

Capítulo 3
Aspectos Metodológicos

La propuesta de un proyecto docente no se limita a la definición de unos contenidos temáticos y a su distribución temporal. Es necesario acompañar estos contenidos de una metodología que permita la consecución de los objetivos planteados. Los avances de las Tecnologías de la Información proporcionan herramientas cuya utilidad mediante un uso adecuado es indiscutible. Estos aspectos son analizados en detalle en este capítulo.

3.1 Introducción

Etimológicamente, la palabra "método" procede de las voces griegas "meta" (*a través de*) y "odos" (*camino*). Literalmente, el método es "*el camino para la realización y cumplimiento de un determinado objetivo*". El término método presenta dos acepciones bien diferenciadas: por un lado sería *el modo de hacer o decir una cosa*; y por otro *el procedimiento que siguen las ciencias para hallar la verdad y enseñarla*. De estas acepciones se puede inferir que **un método de enseñanza** o **una metodología docente** es "*el conjunto de normas y procedimientos destinados a dirigir el aprendizaje de forma eficiente*". A través de él, se debe procurar la correcta ordenación de todos los elementos que integran la acción educativa con el fin de mejorar el proceso e incrementar la seguridad y eficacia del mismo en la consecución de los objetivos establecidos.

El método docente aplicable al desarrollo de una asignatura debe partir siempre de la definición de los objetivos que se persigue alcanzar. A continuación hay que establecer los contenidos y, sirviéndose de los medios o recursos instrumentales disponibles, se determinan las estrategias metodológicas que se van a llevar a cabo. Por último, es preciso establecer los mecanismos de evaluación que permitan determinar en

qué medida se han alcanzado los objetivos propuestos y en qué medida el proceso docente seguido ha sido el más apropiado.

Teniendo en cuenta todos estos puntos se pasa a continuación a describir las peculiaridades del método, que sirve como base para el desarrollo de la actividad docente a la que se refiere este proyecto. En particular, se van a presentar los puntos clave que se han mencionado:

- *Establecimiento de objetivos*
- *Análisis y selección de contenidos*
- *Método y técnicas docentes*
- *Evaluación*

Se ha de tener en cuenta que la descripción concreta de los contenidos y los medios se describirán posteriormente en los capítulos dedicados a los programas de las asignaturas objeto del perfil de la plaza a concurso.

3.2 Ideas básicas de pedagogía y didáctica

Sin pretender ser exhaustivos, el objetivo de este apartado es exponer unas ideas generales sobre la actividad educativa y docente que pasan a detallarse a continuación.

3.2.1 Pedagogía

Sin duda, se está asistiendo en estos momentos a una fase en la que los aspectos o criterios pedagógicos alcanzan cada vez más importancia. Es evidente que un profesor de Informática no tiene que ser un experto en temas pedagógicos, pero como profesional de la docencia sí debe preocuparle, ya que los objetivos educativos, que le corresponden, dependerán en gran medida, de cómo se enseña. "*Descuidar la atención a los métodos con la intención de dedicarse a los contenidos es falso camino; porque los métodos - sin perder su función instrumental - pueden impedir, si no son adecuados, la transmisión de cualquier contenido*" [Gómez, 1981].

La pedagogía moderna ha originado un giro importante en los planteamientos tradicionales de la enseñanza en lo que se refiere a contenidos. En este sentido, se descarta la materia o disciplina de forma aislada y se tiende al desarrollo integral del individuo, buscando su inserción en el contexto social en el que ha de desenvolverse. El objetivo fundamental de este enfoque es la formación integral del alumno, para lo cual deben tenerse en cuenta: *las necesidades de la sociedad y sus recursos*, *los grupos profesionales*, *el progreso científico*, *las aptitudes de los estudiantes*, *el sistema cultural y social...*

Los métodos tradicionales se basan casi exclusivamente en la enseñanza, dando por sentado que la información impartida al estudiante es siempre aprendida. Estos métodos

se preocupan de la forma en que la información es transmitida sin analizar a fondo lo que se aprende, por quién, con qué rapidez, y sobretodo con qué fines, colocando al estudiante en una posición pasiva.

El método de basarse en la enseñanza es el único concebido hasta la aparición de las teorías de **Rousseau**. En palabras de Rousseau en "*Emilio o la Educación*" (libro III): "*Me basta con que sepa encontrar el para qué de todo lo que hace y el por qué de todo lo que cree. Pues una vez más mi objetivo no es darle la ciencia sino enseñarle a adquirirla cuando la necesite, hacerle estimar exactamente lo que vale y hacerle amar la verdad por encima de todo*". Hasta entonces, los objetos importantes eran el saber y el maestro. La innovación de Rousseau y sus sucesores fue simplemente trasladar el fundamento de la ciencia pedagógica desde el *saber y el maestro* al *discípulo*, y reconocer que es el discípulo y sus condiciones peculiares lo único que puede servir de guía para construir una metodología docente.

El aprendizaje es un proceso dinámico de interacción en el cual es primordial el estudiante. Éste no sólo recibe, sino que aporta su contribución. Su percepción de la información es tan importante como la emisión de la misma por parte del docente, y su participación y opinión sobre un programa de formación puede ser más válida que la de los propios docentes. El punto de partida básico debe de ser, por tanto, *la adquisición de conocimientos* y no su transmisión, haciendo que el aprendizaje se base en la satisfacción personal de alcanzar el grado de competencia requerido para el ejercicio profesional.

Es por ello que la pedagogía moderna concede al aprendizaje activo y a la participación del alumno en su proceso de formación un papel fundamental. Ni que decir tiene que el papel del profesor, ante estos nuevos enfoques, se altera de modo sustancial. Se ha de pasar del profesor que posee el saber, toma decisiones, se hace escuchar y explica conocimientos, al profesor que promueve el saber, crea responsabilidades y capacidades para afrontarlas, enseña a tomar decisiones, incita a la participación y, junto a la exposición de sus conocimientos, aplica técnicas de trabajo y de enseñanza. Desde esta óptica, la función del profesor es garantizar que el alumno realice su propio aprendizaje, lo que lleva a un entendimiento de la enseñanza como un proceso activo bidireccional. Esta concepción conlleva un cambio de actitud y de funciones del docente: de la instrucción - transmisión de conocimientos propiamente dichos - se pasa a la investigación y a la discusión. Las aulas se convierten entonces en un centro de trabajo que une al profesor que investiga y enseña, y al alumno que investiga y aprende.

La aplicación de un aprendizaje activo a la enseñanza de la Informática, hoy es algo que parece incuestionable. Una metodología activa que fomente el espíritu de participación, el hábito de estudio, la capacidad de enfrentarse a problemas y decisiones diferentes, la búsqueda de fuentes bibliográficas y otros medios apropiados, la actitud de crítica constructiva..., es algo consustancial a este tipo de asignaturas.

3.2.2 Didáctica

La didáctica es *el conjunto de técnicas a través de las cuales se realiza la enseñanza para que ésta resulte más eficaz*. Se diferencia de la pedagogía en que su ámbito es más reducido que el de esta última, puesto que la pedagogía se ocuparía a la par tanto de la enseñanza como de la educación.

Por tanto, el objetivo principal de la didáctica es orientar la enseñanza mediante un conjunto de procedimientos y normas destinadas a dirigir el aprendizaje de forma eficaz. El marco en el que esos procedimientos y normas se materializan en el desarrollo de enseñanza y aprendizaje constituyen el acto didáctico, en el que intervienen:

- *El sujeto que se instruye, que aprende, el discente.*
- *El sujeto que orienta, que ayuda, que enseña, el docente.*
- *La propia naturaleza del objeto de la enseñanza.*

Estos tres elementos señalan, en cierta manera, el método, el modo y el programa de la enseñanza, indicando también las cualidades que el profesor ha de tener, en cuanto a conocimiento de la materia y para el conocimiento psicológico de los alumnos. Estos conocimientos le permitirán, una vez conocida la realidad de su alumnado, adaptar su enseñanza al mismo, teniendo una visión clara de la finalidad que se persigue. Es por lo que a la didáctica se la ha señalado como una conquista personal, sin normas rígidas, en cuanto que cada profesor tomará las que mejor le vayan con su idiosincrasia y también las que estén más cerca de la realidad de sus alumnos.

El acto didáctico da lugar a un diálogo constante *docente/discente*, a través de la comunicación directa de los contenidos o bien, a través de la orientación al alumno para que él los adquiera por medio de otras fuentes. **A. del Pozo Pardo** [Pozo, 1982] distingue las siguientes características del acto didáctico, sobre la base de las consideraciones anteriores:

- Se trata de una actividad coordinada y conexionada que implica a ambos agentes (*discente y docente*) y supone una comunicación entre ambos.
- Es una asociación intencional y consciente que conlleva una decisión consistente en un propósito de enseñar por parte del docente y un propósito de aprender por parte del discente.
- Persigue la consecución de dos objetivos: *la enseñanza* y *el aprendizaje*.

Siguiendo al autor citado, **A. del Pozo Pardo** [Pozo, 1982], esta interrelación de elementos se materializa en el acto didáctico a través de tres actividades, éstas son la enseñanza, el aprendizaje y la planificación:

- **Enseñanza**: El docente se relaciona con el discente y la materia que debe impartir mediante la actividad considerada como la enseñanza,

transmitiendo nociones, conocimientos, habilidades... o posibilitando al discente el acceso a ellas: *buscando el máximo desarrollo de sus facultades, hábitos y conductas*. La eficacia del docente depende de dos factores fundamentales: *conocimiento profundo de la materia*, y *arte o técnica de enseñarla*. El primero exige un continuo contacto con las fuentes de erudición, una puesta al día y una profunda reflexión sobre la disciplina. El segundo requiere no sólo el conocimiento de las técnicas pedagógicas, sino su continua puesta en práctica con espíritu crítico sobre la propia tarea.

- **Aprendizaje**: El discente se relaciona con el docente a través de la materia objeto de la enseñanza, dando lugar a la actividad del aprendizaje. Recibidos los contenidos o el estímulo del docente, el discente los asimila, posibilitando su posterior utilización. En este sentido, se hace fundamental considerar el nivel intelectual y humano de los discentes, teniendo en cuenta que se hallan en un período de formación, tanto académica como de su personalidad.
- **Planificación**: El docente se relaciona con la materia que debe impartir a través de la planificación o programación, actividad ésta que le es exclusiva y mediante la cual formula los objetivos que pretende conseguir. En esta actividad selecciona, analiza y ordena los contenidos que impartirá, delimita la metodología que se aplicará y por último, elige los recursos y medios más adecuados para lograr la máxima eficiencia en su labor docente. Una depurada selección de materias, prescindiendo de los temas menos importantes, junto a la elaboración de un programa lógico y coherente, que no excluya la profundización en aquellos aspectos que se consideren necesarios, son requisitos fundamentales para el docente respecto a la materia que va a enseñar.

La planificación conduce a preguntarse cómo transmitir los conocimientos. En palabras de **Lafourcade** [Lafourcade, 1974], "*uno de los supuestos claves que contribuyen al logro de una enseñanza de calidad, es la preparación de un plan de acción que articule, de modo racional, los diversos componentes de la tarea didáctica que se debe cumplir*". Podría afirmarse que el nivel de los rendimientos que logran los alumnos, es una resultante directa del tipo de estrategia, que se haya planteado y de los modos que se hayan seleccionado para llevarla a la práctica. A modo de síntesis, podemos señalar que cualquier plan de acción ha de recorrer una serie de fases o etapas que nos permitan responder por este orden a las siguientes preguntas:

- *¿Qué se pretende con la asignatura y con sus contenidos? Es decir, sus objetivos.*
- *¿Con qué medios se cuenta?*

- *¿Qué actividades y experiencias de aprendizaje deben realizarse para alcanzarse esos objetivos?*
- *¿Cómo organizar y secuenciar estas experiencias y actividades para facilitar su asimilación en el alumno?*
- *¿Cómo conocer si se han alcanzado los objetivos propuestos? ¿Cómo evaluar la eficiencia de esas actividades en función de los objetivos?*

De donde, en principio, se podrían establecer tres etapas para llevar a cabo el proceso didáctico: Programación, Ejecución y Control. En la Figura 3.1 se muestra un esquema de estas etapas junto con los factores que influyen en cada una de ellas y que se explicarán más adelante.

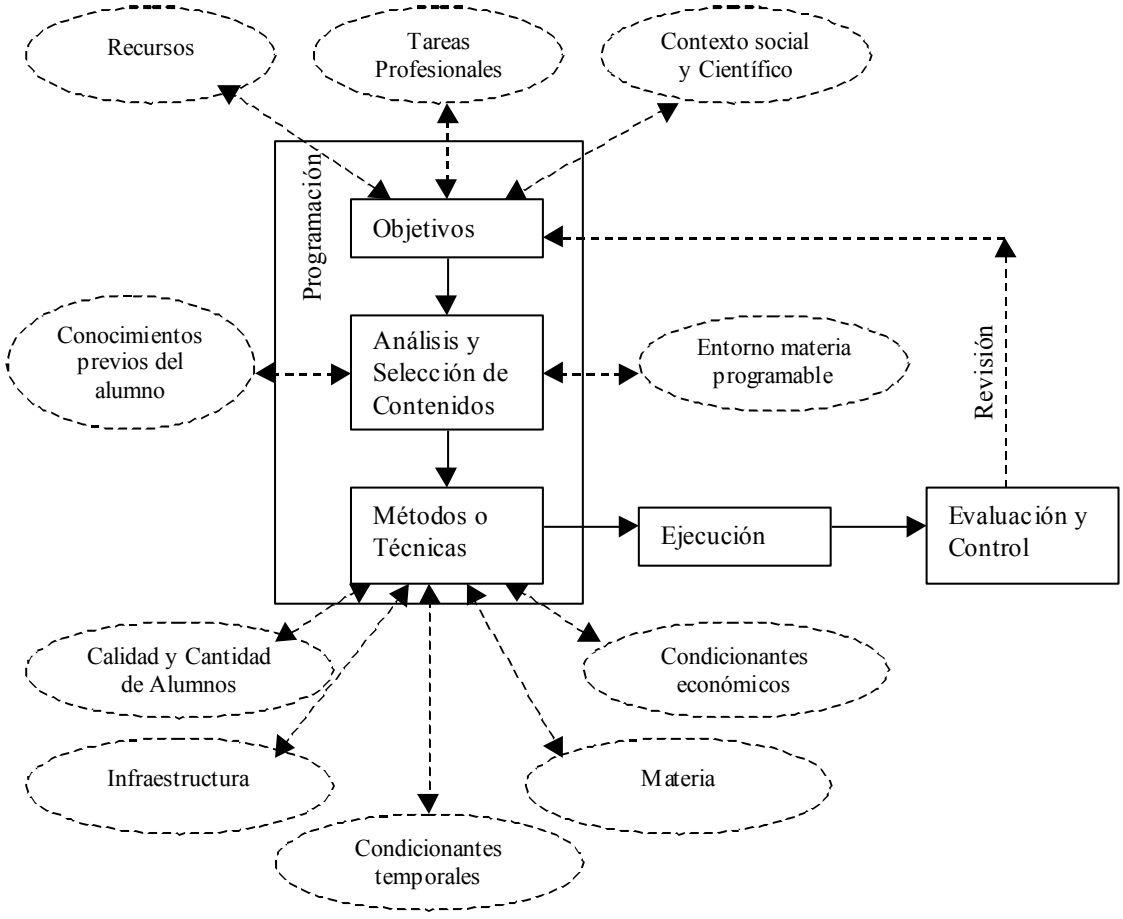

Figura 3.1. Etapas de proceso didáctico

La primera etapa representa el diseño de la intervención y supone el establecimiento de los objetivos, el análisis y selección de los contenidos, y la elección de los métodos o técnicas aplicables. La segunda fase consistirá en ejecutar lo programado, y la tercera en analizar la eficacia, comportamiento y relevancia de todas las variables, así como de las posibilidades mostradas por las estrategias docentes y del grado de asimilación y efecto provocado en el alumno.

El concepto de programación es uno de los logros de la moderna tecnología educativa y se fundamenta en la convicción de que la educación, entendida como tarea racional y sistemática, exige saber, previamente, de manera precisa y concreta, qué objetivos deben alcanzar los alumnos, puesto que aquellos serán, en definitiva, los que objetivan los medios para conseguirlos y los criterios que se emplearán en su valoración.

Según lo expuesto, el concepto de programación sería un proceso que coordina objetivos, medios y criterios de medida. O dicho de otra forma, la programación es un proceso abierto, continuo y en permanente transformación, a través del cual se confecciona un programa, siendo este último "*todo intento de enunciar por escrito lo que se va a hacer*" [Pozo, 1982].

Es frecuente utilizar indistintamente los términos programar y planificar, sin embargo, existen matices que los diferencian. Estos matices estarían basados en la mayor o menor concreción y precisión de los aspectos previsibles. La planificación será más general y a más largo plazo; la programación, en cambio, será más específica, más concreta y a corto plazo. De ambas tareas se deberá ocupar el profesor, de un lado tendrá que elaborar una programación a corto plazo y de otro tendrá que planificar las actividades que se realizarán a lo largo de los cursos con el fin de conseguir los objetivos propuestos.

Tanto se trate de planificación como de programación, el primer punto a definir en ambos casos será un **conjunto de objetivos**. Los objetivos se pueden definir como *el aprendizaje de nuevas capacidades que una institución se esfuerza por obtener en sus estudiantes*.

Una vez definidos los objetivos educativos se debe planificar **un programa de formación**, es decir, seleccionar una serie de actividades encaminadas a conseguir el logro de los objetivos. Como paso final, se hace necesario definir **los procesos de evaluación**. Éstos deben permitir valorar en qué grado se han alcanzado los objetivos propuestos a través del programa de formación concreto, permitiendo la medida de la competencia final de los educandos, así como la determinación de la eficacia de programas y docentes.

La evaluación del programa va íntimamente ligada a su evolución. Es por ello que la programación nunca es algo totalmente acabado, sino que tiene un carácter dinámico, ha de estar en permanente revisión en función de los resultados obtenidos. Por tanto, la elaboración y modificación del programa ha de tener presente el entorno en el que esté la materia programable, tanto a nivel global como particular en lo que se refiere al Centro en el que se imparte y a las relaciones que se pudieran establecer con el resto de las disciplinas de la titulación.

Adicionalmente, los nuevos elementos que llegan al Centro desde el contexto social y científico pueden ocasionar la revisión de los objetivos y en consecuencia del

programa. Esto se hace particularmente patente en el ámbito de la docencia en Informática, dada la rápida evolución tecnológica a la que se ve sometida.

A continuación se van a tratar de forma individual las tres etapas para realizar la fase de programación: *objetivos, elaboración del programa de formación* y *los métodos o técnicas para evaluar el proceso formativo*.

3.3 Los objetivos educativos

Los objetivos educativos son definidos por **Gullbert** [Gullbert, 1989] como: "*Lo que es necesario que los estudiantes sean capaces de llevar a cabo a la terminación de un período de enseñanza/aprendizaje y que no eran capaces de realizar antes*". Por tanto puede decirse que lo que se persigue, mediante la acción del sistema educativo, es que el alumno adquiera algo que no tenía. Éste sería en definitiva el objetivo básico, que puede enunciarse como el enriquecimiento que se pretende implantar en el alumno.

Ningún sistema de enseñanza puede ser eficaz si no se fijan sus objetivos con claridad. Dentro del proceso educativo los objetivos representan el punto de partida y la referencia para la valoración de su eficacia, ya que con ellos quedan descritos con precisión los resultados a lograr. Es a partir del establecimiento de los mismos cuando el docente se encuentra en condiciones de seleccionar los contenidos a impartir, los medios a emplear y las estrategias didácticas y metodológicas a utilizar.

Del análisis de los objetivos se pueden extraer tres rasgos o características esenciales, que se desprenden de todas aquellas definiciones de objetivos que se planteen. Según **J. Rodríguez Dieguez** [Rodríguez, 1980]: "*los objetivos son planteados, (1) poniendo como sujeto al alumno, (2) se enuncian como resultados futuros, y (3) se pueden observar y evaluar*".

El establecimiento de los objetivos con precisión tiene múltiples ventajas, ya que:

- Facilita la evaluación de las metodologías.
- Orienta al alumno en su proceso de aprendizaje.
- Sirve como medio para la autoevaluación del profesorado.
- Incrementa la motivación de los alumnos, al conocer con precisión lo que se les va a pedir.
- Facilita la calificación, ya que favorece la objetividad.
- Contribuye a la elección correcta de los métodos didácticos.
- Posibilita una mayor coordinación entre profesores.

Por otro lado, la formulación de los objetivos no está exenta de pocas dificultades, entre las cuales resaltan:

- El hecho de que todos los objetivos no se pueden explicar. A menudo, se suscitan en el aula cuestiones que no figuran en el programa, por lo que se formularán tan solo los objetivos más relevantes, aquéllos que dan sentido de dirección al proceso de enseñanza.
- El problema de la especificación de los objetivos no se circunscribe a una cuestión meramente técnica, sino que implica una visión personal de la educación.

Es por ello que debe de cuidarse la corrección en la formulación de los objetivos. Para que el enunciado de un objetivo sea correcto debe de poseer las siguientes características:

- **Prospectivos**, ya que se forman profesionales para el futuro.
- **Pertinentes**, por su correlación con las tareas profesionales y la realidad social.
- **Precisos y concretos**.
- **Realizables**, contando con los recursos existentes y, principalmente, con la capacidad de aprender del alumno, condicionada tanto por los conocimientos previos que le hayan sido impartidos como, por la cantidad de información que éste pueda asimilar.
- **Mensurables**, para que mediante la técnica adecuada de evaluación pueda conocerse en qué grado se han conseguido.

De todas estas características la más esencial es la pertinencia. Cualquier objetivo que reúna todas las demás y no sea pertinente es potencialmente peligroso desde el punto de vista docente.

Los objetivos son de diversa índole, distinguiéndose entre *objetivos de carácter formativo del individuo como tal* y *objetivos específicos*, propios de la naturaleza de la materia a impartir. A continuación se desarrolla esta clasificación con mayor profundidad.

3.3.1 Objetivos formativos de tipo general

Desde un punto de vista general se pueden identificar en la literatura objetivos de formación que se podrían decir que son válidos para cualquier disciplina. Así, **Bloom** [Bloom, 1956] enuncia una taxonomía que se encarga de enumerar y definir los objetivos de la acción educativa. Dicha taxonomía propone una serie de objetivos, algunos de ellos se refieren a hábitos no meramente intelectuales. Del trabajo de Bloom se observa que los objetivos educativos pueden dividirse en tres grandes bloques o categorías:

- **Objetivos cognitivos**, de desarrollo de la capacidad de recuerdo de datos, de su interpretación y de resolución de problemas;
- **Objetivos psicomotores**, o de adquisición de las aptitudes que requiere la metodología propia de cada disciplina educativa;
- **Objetivos afectivos**, o de motivación del alumno, desarrollando su receptividad y su capacidad de respuesta ante los problemas o situaciones planteadas.

En el caso concreto de la enseñanza universitaria, y más aún de la de la enseñanza técnica, son los objetivos cognitivos los que presentan una importancia mayor. Sin embargo, no conviene olvidar aspectos como la motivación, que en ocasiones pueden resultar fundamentales a la hora de transmitir los conocimientos y de lograr no sólo el aprendizaje, sino la formación integral que la LRU defiende.

Siguiendo con la referencia al trabajo de Bloom, los objetivos cognitivos se clasifican a su vez en las siguientes categorías:

- **Conocimiento**. El alumno debe aprender una serie de datos específicos, principios, métodos, criterios, clasificaciones..., dentro del campo de estudio. A partir de los conocimientos debe adquirir la capacidad de enunciar, enumerar, describir o definir.
- **Comprensión**. El aprendizaje no puede ser puramente memorístico, el alumno debe comprender las relaciones existentes entre los distintos elementos, diferenciar lo accesorio de lo fundamental y ser capaz de interpretar los conocimientos adquiridos.
- **Aplicación**. Conocer y comprender un determinado método puede no ser suficiente para saber aplicarlo. Efectivamente, la aplicación lleva consigo un grado de abstracción más elevado; al enfrentarse a un determinado problema el alumno debe recurrir a unos ciertos procesos mentales para intentar reducirlo a una situación que pueda ser resuelta mediante la utilización de abstracciones conocidas. Sólo si lo consigue se puede decir que ha adquirido la capacidad de aplicación.
- **Análisis**. El análisis se refiere a la capacidad del alumno para dividir un todo en sus partes fundamentales, obtener sus interrelaciones y su modo de organización.
- **Síntesis**. Es el proceso inverso, que supone, por ejemplo, que el alumno sepa extraer información de diversas fuentes y sea capaz de organizarla de un modo general para elaborar un nuevo material. Este proceso supone el desarrollo de conductas creadoras, es un proceso constructivo y presenta también una gran importancia.

- **Evaluación**. Otra de las conductas básicas que el alumno universitario debe desarrollar es la actitud crítica, es decir, la capacidad de valorar las ideas, los métodos...

La educación es un proceso cuyo fin esencial es modificar la conducta en cuanto a conocimientos, actitudes y aptitudes. La consecuencia de dicha modificación debe ser la aportación al educando de ideas, actitudes, hábitos e intereses que no tenía antes. Por ello, en el ánimo de completar la taxonomía anteriormente presentada, se pueden añadir los siguientes objetivos generales:

- *La capacidad de expresión oral y escrita*. El saber sintetizar las conclusiones de un trabajo. Exponerlas tanto en documentos escritos como en exposiciones audiovisuales ante el público.
- *El respeto por valores de tipo humanístico y unos principios éticos asociados a la profesión*.
- *El interés por la exactitud y rigor*.
- *La capacidad para cooperar con otros profesionales*. Saber trabajar en grupo. Coordinar y planificar tareas.

3.3.2 Objetivos específicos

Ahora bien, preservando estos objetivos de carácter general, no se deben de descuidar aquellos de carácter específico que estén asociados a la materia que se pretende abordar.

Como se ha señalado anteriormente, se admite que en cualquier materia existen tres taxonomías o campos [Bloom, 1956]: *cognoscitivos, afectivos* y *psicomotores*. Por tanto, el enriquecimiento que deben de proveer unos objetivos educativos en una materia concreta, consiste en los conocimientos que se deben adquirir y también, de las habilidades que se deben desarrollar.

En el ámbito universitario significa proveer al alumno, por una parte, de los conocimientos especializados (*finalidad material de la didáctica*) y, por otra, de una educación de base más general, que le amplíe las perspectivas sobre el mundo y sus problemas (*finalidad formal de la didáctica*) como dice **Gullbert** [Gullbert, 1989]. A través de este último cauce es posible enunciar una serie de objetivos que, con matices, es común a la mayoría de disciplinas, a saber:

- Que los conocimientos expuestos aporten al futuro titulado la terminología y las leyes principales de una materia, así como la familiarización con las principales aplicaciones de la misma. De tal forma que entienda, aprenda y asimile los conceptos básicos, ideas fundamentales y datos específicos, permitiéndole enfrentarse, aplicando los conocimientos adquiridos, con la resolución de casos prácticos similares a los que se le plantearán en su vida profesional.

- Manejar bibliografía, hacer que el alumno disponga de la suficiente información y bibliografía que permita ampliar sus conocimientos en un determinado tema cuando sus necesidades lo requieran, buscarla bien en soporte papel o magnético (ayudas en línea, Internet).
- La capacidad no sólo de comprender la extensión y significado de lo que ya se conoce en el campo donde se encuadra la disciplina, sino de ser receptivo ante lo nuevo, de afrontarlo y de trabajar confiadamente de forma personal. Según **Fernando Lara** [Lara, 1997] "*Uno de los objetivos imprescindibles debería de ser siempre preparar al alumno para que pueda aprender y para que desee aprender nuevas cosas relativas a la asignatura*".

Estos dos últimos puntos tienen especial importancia en el campo de la Informática. El vertiginoso avance de las ciencias de la información obligan a que los profesionales estén sometidos a un proceso de autoformación continua, debiendo de esclarecer lo que en la actualidad es válido e innovador, de lo que pueda quedar obsoleto.

Para definir objetivos más específicos de la materia que se esté impartiendo, se deben conocer las necesidades y posibilidades de los alumnos a los que va dirigida la enseñanza, las realidades sociales y los recursos con los que se cuenta para la educación, además del conocimiento claro y preciso de los fines que se deben alcanzar, de modo que a la finalización del proceso, los alumnos estén especialmente preparados para el ejercicio de su profesión.

Consecuentemente, el primer paso que se debe dar en una planificación educativa es definir correctamente las tareas profesionales. Esta definición debe derivar del estudio de las necesidades sociales para este tipo de profesionales, y saber de modo claro y preciso lo que las personas tendrán que hacer en el curso de su ocupación. Si los objetivos se elaboran de este modo, cuando la sociedad evolucione y sus necesidades y tareas profesionales cambien, los objetivos educativos lo harán en consecuencia. Si por el contrario se deja de lado la definición de las tareas profesionales, podría suceder, como indica **R. F. Mager** [Mager, 1979], que "*si no se está seguro de a donde se quiere ir, se corre el riesgo de encontrarse en otra parte y no darse cuenta*".

Para la determinación de dichas tareas profesionales, existen distintos métodos de los cuales los más importantes se introducen brevemente a continuación.

- **Técnica del Incidente Crítico**. Basado en estudiar para una situación o incidente determinado, cuáles han sido los aciertos y errores, así como los factores fundamentales que han podido contribuir ha dicho éxito o fracaso. Esto puede dar orientaciones al respecto de la trascendencia de determinados conocimientos, actitudes o aptitudes. En este sentido el incidente crítico puede venir dado muchas veces por la propia experiencia profesional del docente. Por ello, se hace muy interesante que el profesor haya sido en algún momento por un lado discente de la materia que está

impartiendo, así como que por otro lado, haya intentado ponerla en práctica como profesional.

- **Técnica del Análisis de Funciones**. Consiste en estudiar las competencias de la actividad profesional sobre la que se pretende impartir docencia. Esta técnica exige que este análisis venga dado a través de una institución representativa o de prestigio, de lo contrario es fácil caer en conclusiones subjetivas. Existen distintos modelos de currículos propuestos por entidades de renombre que serán presentados en secciones posteriores.
- **Estudio de la Demanda Laboral**. El estudio de la demanda laboral se puede llevar a cabo por un lado a través de un estudio de dicha demanda expresada a través de los medios de comunicación. Un ejemplo de este tipo de información se puede ver en [González, 1996][6]. Este estudio es fácil de obtener y muy útil de cara a identificar los elementos informativos, no así tanto los formativos que pueden requerir el contacto directo con empleadores y responsables de recursos humanos de organizaciones. Dichos responsables deben, a ser posible, constituir una muestra con una heterogeneidad representativa del entorno real de trabajo, lo que permitiría llegar a unas conclusiones válidas.
- **Encuestas Profesionales**. Este tipo de encuesta es difícil de obtener en cuanto que debe de haber organismos profesionales encargados de llevarlas a cabo. En el caso que nos compete las únicas encuesta a las que se ha tenido acceso son los trabajos de [McLeod, 1996] y [Sanchís y Torralba, 1997].

La combinación de todos estos métodos permite un mayor acercamiento a la realidad del momento. Así mismo, la previsible variación de algunos de estos datos de entrada hace recomendable la revisión periódica de los objetivos. En todo ello está de forma implícita la aceptación, por parte de las personas encargadas de la docencia, de una mayor responsabilidad que lo que supone la mera preparación de profesionales que se adapten a la situación del instante presente: unos buenos objetivos docentes deben definir también las futuras necesidades y tendencias de la profesión, y preparar personas capaces de ajustarse a ellas.

3.4 Análisis y selección de contenidos

Los contenidos son el instrumento que va a permitir alcanzar los objetivos formulados. En este sentido se entiende que el contenido es un medio para algo, y no un fin en sí

[6] Aunque este estudio es muy exhaustivo se centra fundamentalmente en la demanda de titulados superiores y medios en informática en relación a los no titulados, por lo que no será válido para determinar objetivos de asignaturas.

mismo. La selección de contenidos no es entonces arbitraria, sino que está determinada por una serie de factores, entre los que destacan:

- *Los objetivos generales de la carrera y los específicos de la asignatura.*
- *La estructura de la materia.*
- *Los planes de estudios.*
- *El contexto académico.*
- *El contexto socio-profesional.*
- *Los conocimientos previos de los alumnos.*
- *Los contenidos de los programas de asignaturas relacionadas con la materia.*
- *La experiencia del profesor.*

A la hora de establecer los contenidos más adecuados ha sido preciso tener en cuenta unos criterios básicos de selección, como pueden ser los siguientes:

- *Congruencia con los objetivos previstos.*
- *Identificación de los núcleos básicos de la materia.*
- *Representatividad dentro del amplio conjunto de conocimientos.*
- *Posibilidad de transferencia entre los conocimientos proporcionados por la misma u otras materias.*
- *Validez y permanencia en el tiempo.*
- *Actualidad, para no perder de vista la evolución a la que puedan estar sometidos por el desarrollo de la técnica, la ciencia o la sociedad.*
- *Graduación secuencial, para avanzar de lo más simple a lo más complejo.*

En este sentido, se han definido los contenidos teóricos de las asignaturas de objeto de este proyecto docente siguiendo de forma importante los criterios que se acaban de señalar y las limitaciones existentes.

3.5 Métodos y técnicas

Una vez establecidos los criterios a tener en cuenta en la definición de los objetivos educativos y los contenidos que se incluyen en el proyecto, es necesario determinar que actividades de aprendizaje o métodos de enseñanza son los más idóneos para conseguir cada uno de los objetivos. En esta sección se pretende ver cómo se puede elaborar el programa de las materias objeto de este proyecto, tanto desde una perspectiva metodológica, como desde el análisis de las técnicas a disposición del docente para enfrentarse a su trabajo.

3.5.1 Criterios metodológicos en la elaboración del programa

Hay ciertos principios metodológicos básicos observables para la consecución del aprendizaje de los alumnos en las materias de estudio, así como para que éste sea exitoso. Estos principios pueden resumirse en los siguientes:

- *Continuidad*
- *Progresión continua de dificultad*
- *Dignidad en los contenidos y en su presentación*
- *Posibilidad de revisión*
- *Realismo en los contenidos*
- *Diversidad en la presentación*

3.5.1.1 Continuidad

Evitar, en lo posible, que partes de la materia queden como compartimentos aislados sin relación con el resto; hay que fomentar una visión integradora e interdisciplinaria de los conocimientos científicos, posibilitando el ejercicio de las habilidades de síntesis. Este criterio implica la ausencia de saltos, lagunas o, lo que es peor, incongruencias en la materia que se imparte y que afecta, de manera directa, al planteamiento del programa de las asignaturas.

3.5.1.2 Progresión continua de dificultad

Conviene llevar al alumno siempre un paso por delante de su capacidad actual, pero no más. Así, el elemento motivador de superar las dificultades mediante el esfuerzo, no se convertirá en un muro de apariencia infranqueable que lo desanime. Este criterio entraña conflictos con el método habitual de ir desde lo que es general a lo que es particular, ya que es frecuente que lo general entrañe mayor dificultad. Sin embargo, progresar de lo general a lo específico facilita la visión global del tema y ahorra tiempo en el aprendizaje.

En cada caso, hay que decir si es conveniente o no el empleo de esta técnica basándose en las características de los contenidos docentes particulares.

3.5.1.3 Dignidad de los contenidos y en su presentación

La necesidad del establecimiento de este punto es obvia para conseguir la competencia del alumno como profesional. A parte de esto, se considera una reacción normal del alumno la pérdida de interés por la asignatura cuando, por cualquier motivo, llega a juzgar sus contenidos como carentes de utilidad o como presentados de forma inadecuada. Por tanto, han de estar diseñados de forma motivadora y en clara relación con la realidad. En este sentido la planificación de los contenidos ha de ser:

- **Colectiva**: elaborada mediante trabajo en equipo.
- **Completa**: debe de dar un enfoque sistemático que interrelacione todos los elementos del programa.
- **Concreta**: los elementos estructurales básicos de un programa pertinente deben de ser las tareas profesionales concretas perfectamente definidas.
- **Integrada**: tanto por la inclusión de aspectos teóricos y prácticos, como por su coordinación con la programación de otras asignaturas, evitando la reiteración de materias o exposiciones incomprensibles por falta de conocimientos previos.
- **Motivadora**: la enseñanza es un proceso activo más eficaz cuanto más motivado esté el estudiante para aprender. Es importante planificar una enseñanza que sea capaz de despertar actitudes e intereses. Si la forma de enseñanza sólo se preocupa de desarrollar conocimientos y habilidades técnicas, por muy completa que sea la práctica de estos aspectos, nunca podrán sustituir a las actitudes y los intereses, es decir al compromiso personal del estudiante.

Conviene, especialmente tratándose de alumnos de cierta madurez, como es el caso, incluir breves reseñas sobre la conveniencia de la metodología empleada y del enfoque presentado. Esto, junto con la credibilidad que proporciona al alumno la comprobación de la capacidad del profesor, la cual constata diariamente, será suficiente para evitar la desmotivación en el sentido apuntado.

Se da por supuesto que lo anterior implica una labor de convencimiento del alumno, para lo que el profesor debe dotarse de argumentos convincentes, siendo el contenido del programa uno de los fundamentales.

3.5.1.4 Posibilidad de revisión

Hay que considerar, en todo momento, al contenido como un bien cultural susceptible de adecuación al momento que se vive. Este punto es especialmente notorio en lo concerniente a la docencia en Informática, debido a su rápida evolución.

Por tanto se ha de recoger tanto tendencias y líneas de investigación, como sobre todo aspectos de aplicación sobradamente extendidos en ese momento, revisando los contenidos para restar importancia a aquellos que queden obsoletos.

3.5.1.5 Realismo en los contenidos

Los contenidos del programa deben, por un lado, ser adecuados al nivel de conocimientos que tienen los alumnos y a la utilidad de éstos para el desempeño futuro de su profesión.

Señalar la gran dificultad que existe, al tratar de elaborar el programa, por las diferencias de nivel de conocimientos entre los distintos alumnos originada por la diferente formación de los mismos (*Formación Profesional, COU, LOGSE*). Por otra parte, hay que procurar que sus contenidos se ajusten al tiempo real de que se dispone para su desarrollo.

3.5.1.6 Diversidad en la presentación

Contemplar métodos pedagógicos diversos, de manera que el resultado sea un conjunto equilibrado, tanto en cuanto a las diferentes técnicas utilizadas, como a la aplicación de cada una de ellas en cada momento.

También es interesante la alternancia de las cuestiones teóricas con ejercicios u otras prácticas que den lugar a una visión más pragmática de la asignatura y faciliten la comprensión de sus contenidos.

En otro orden de cosas, es interesante recordar los tres tipos de enseñanza que existen según la metodología que se siga en el proceso de aprendizaje. Dichos métodos se deben considerar a la hora de establecer los contenidos del programa para combinarlos de forma adecuada. Estos métodos son los siguientes:

- **Didáctico**: Siguiendo este método, el profesor explica a los alumnos la realidad objetiva u objetivada que se supone posee y que es transmitida al alumno en el acto docente. Éste recibe de las clases más información que formación, privándole, por tanto, del necesario proceso de deducción. El método didáctico tiene el inconveniente de que el alumno se ve abocado a una excesiva memorización debido a que no deja mucho espacio para su participación, pero, de otro lado, tiene la ventaja de que permite al profesor programar la enseñanza adaptándola al tiempo disponible para su desarrollo.
- **Dialéctico**: Implica la búsqueda de la verdad mediante el contraste de opiniones y enfoques distintos. En esta dialéctica, el profesor es el que tiene mayor responsabilidad y debe dirigir la discusión hacia los puntos de interés, pero con habilidad suficiente para que las conclusiones aparezcan como fruto de la discusión y del razonamiento en común. El profesor requiere de una gran capacidad de improvisación y asimilación que le permita mantener el tema dentro de los límites sustanciales sin que derive hacia cuestiones secundarias que, espontáneamente, surgen en el debate. Por tanto, este método exige de aquél un mayor esfuerzo y una adecuada preparación, así como un buen dominio de las materias, de forma que pueda hacer frente a cuestiones inesperadas sugeridas por los alumnos.
- **Heurístico**: Aquí el alumno es el que debe redescubrir (o descubrir, lo que también pudiera suceder) las soluciones por su cuenta, valiéndose de los conocimientos que ya tiene, realizando así un proceso de autoformación.

Los temas que se tratarán son distribuidos entre los alumnos, pudiendo éstos agruparse o trabajar individualmente. La actividad del profesor es esencial para que el alumno no desvíe su atención hacia temas de su interés dejando sin actualizar aspectos relevantes. Para ello el profesor debe de controlar a los alumnos mediante una adecuada asignación de funciones a los mismos, así como la implantación de las directrices que deben seguir para el desarrollo de su trabajo.

3.5.2 Metodologías docentes para la ejecución del programa

Los métodos de docencia universitaria son los instrumentos por los que el profesor traslada los contenidos de una materia hacia el alumnado, con el fin de conseguir los objetivos propuestos. Por ello, no debe existir una disociación entre los contenidos y los métodos, ya que estos últimos tienen una función destacada en la configuración de los contenidos, por tanto, es un paso obligado en todo proyecto docente analizar los distintos métodos de enseñanza y su aplicabilidad en la materia correspondiente.

La aplicación de los diferentes métodos pedagógicos - didáctico, dialéctico y heurístico -, ya comentados anteriormente, a la enseñanza en las materias objeto de este proyecto pueden desarrollarse, básicamente, a través de clases teóricas, clases prácticas, seminarios y horarios de apoyo a los alumnos o tutorías.

Por tanto, en este punto se expone la forma en que se propone llevar a cabo las diversas actividades de la función docente. Se aborda también las cuestiones de método que son de aplicación en ámbitos distintos al de la elaboración del programa. No obstante, hay cuestiones de método que afectan tanto al programa como a otros aspectos de la función docente y, también, serán considerados aquí.

Actualmente prima la innovación en las técnicas docentes. En este sentido **Fernando Lara** [Lara, 1997] distingue en función de las técnicas docentes dos tipos de enseñanza:

- "*Una enseñanza en que el profesor, desde 'la tarima' transmite la ciencia; y el alumno, desde 'el asiento' la recoge en sus apuntes para estudiarla y para soltarla en el examen, quizás también para entenderla*".
- "*Una enseñanza en que el profesor, además de transmitir la ciencia, desea provocar el aprendizaje posterior del alumno, transmitiendo el interés personal por la asignatura, motivándole a seguir investigando sobre la materia, motivándole a formular preguntas que aclaren lo que no entiende, etc*".

La pedagogía moderna, junto con la tecnología educativa, ofrecen en la actualidad una amplia gama de técnicas y recursos para el desarrollo del proceso enseñanza-aprendizaje. En general, se puede decir que cualquier técnica o método es válido

siempre que se adapte a la actividad programada, a los objetivos propuestos y a las circunstancias. En realidad, método existe siempre; se trata de elegir el mejor para cada circunstancia, sólo así los contenidos serán transmitidos con el mejor nivel de eficacia y rentabilidad respecto a la inversión educativa a que se refiere. Como indica la **UNESCO** [UNESCO, 1973], "*la educación debe poder ser impartida y adquirida por una multitud de medios, ya que lo importante no es saber qué camino ha seguido el sujeto, sino lo que ha aprendido y adquirido*". El fin, y no el medio es, en consecuencia, el punto de mira esencial. Pero ello no implica, ni mucho menos, que el medio sea indiferente, como quizás pudiera deducirse de modo erróneo de las frases de la UNESCO, sino por el contrario:

- *Es el objetivo perseguido el que condiciona la técnica más adecuada.*
- *Las técnicas no son buenas ni malas, sino que pueden estar bien o mal aplicadas en relación con el propósito que se presenta, en función del espíritu que las impregne y, en la medida en que se apliquen de un modo activo, propiciando el ejercicio de la reflexión, el espíritu crítico del alumno...*
- *Si bien pueden existir varias técnicas aplicables a un objeto educativo concreto, siempre podrá aludirse a técnicas más apropiadas y a otras menos adecuadas.*

Por tanto, la ejecución de la acción docente puede ser albergada por diversas técnicas o escenarios, cada uno de los cuales puede llegar a ser apropiado en mayor o menor grado para cada punto concreto del programa. Es por tanto, tarea del profesor distribuir en estos escenarios la materia, tomando como criterio de decisión los medios, la calidad y cantidad de alumnos, los condicionantes temporales y económicos, y la medida subjetiva de la pertinencia de la aplicación de un determinado método docente a una determinada materia.

A fin de referenciar cada una de las actividades docentes a lo largo de la exposición de los programas de las asignaturas, se procede a enumerarlas:

1. ***Clase teórica o lección magistral.***
2. ***Clases de problemas.***
3. ***Clases prácticas.***
   - ***3.1. Prácticas guiadas.***
   - ***3.2. Prácticas libres.***
4. ***Actividades docentes complementarias***
   - ***4.1. Seminarios y conferencias.***
   - ***4.2. Visitas y prácticas en instalaciones y centros profesionales.***
   - ***4.3. Tutorías.***
   - ***4.4. Internet como vía de comunicación con los alumnos.***

A las técnicas mencionadas se han de añadir (*cualquiera que sea la modalidad escogida*), aquellas actividades complementarias que, para un acto concreto, se consideren necesarias, tales como lecturas complementarias, búsqueda de bibliografía, confección de trabajos, asistencia a conferencias...

Antes de iniciar el desarrollo de cada una de las técnicas o modos con los que ha de impartirse el conocimiento, se comentan los principios que han de tenerse en cuenta cara a su elección:

- *Se adecuarán a los objetivos pretendidos en cada una de las partes del programa, lo que implica la selección de diferentes medios o recursos para los diferentes contenidos.*
- *Del punto anterior se desprende que la congruencia entre medios y fines es algo que se debe tener siempre presente.*
- *Que transmitan no sólo conocimientos sino también procedimientos, esquemas de razonamiento, mecanismos de aplicación, generalización y síntesis y, en definitiva, metodología científica.*
- *Que permitan al alumno desempeñar un papel activo (documentarse, exponer, observar, participar...).*
- *Que le estimulen la necesidad de aprender y la iniciativa para la aplicación de sus conocimientos a problemas reales.*
- *Que presenten una cierta flexibilidad para que el alumno pueda tomar decisiones razonables respecto a cómo desarrollarlas.*
- *Que fomenten tanto el trabajo individual como en equipo.*
- *Que puedan ser cumplidas por la gran mayoría de los alumnos, teniendo en cuenta sus diversos niveles de capacidad y sus diferentes intereses.*
- *La selección debe ser equilibrada, repartiéndose a lo largo del curso de forma ponderada. Resulta muy arriesgado realizar estimaciones con carácter fijo sobre lo que cada técnica debe ocupar en el desarrollo del programa.*

En todo caso, se ha de tener en cuenta que la adecuada combinación de estas técnicas ha de contribuir a la creación de actividades críticas, reflexivas y analíticas en los alumnos; de tal forma que puedan llegar a obtener una visión equilibrada e integradora de los distintos aspectos que conforman la realidad estudiada. Se trata, pues, de incorporar una **metodología activa, globalizadora y participativa** en la medida en que las condiciones en las que el desarrollo de la actividad docente, lo permita. De ahí que, muchas veces, no se pueden hacer extensivas estas técnicas de trabajo a la totalidad de alumnos, llegando únicamente a aquellos que, encontrándose motivados, acepten las

mismas voluntariamente, de tal forma que, con este punto de partida, el éxito de las mismas esté casi garantizado.

En general, dentro de las actividades que debe realizar el profesor se suelen considerar tres categorías básicas: *explicación, motivación* y *orientación*.

La ***explicación*** es la base de la transmisión de los conocimientos; aunque en ella es el profesor el que realiza la parte más activa, hay que intentar evitar que el alumno se sienta como elemento meramente pasivo.

La ***motivación*** tiene una gran importancia puesto que influye fuertemente en la capacidad receptiva del alumno. De poco sirve realizar un gran esfuerzo en que la transmisión sea correcta si falla la recepción. Favorecer este aspecto es por tanto esencial para el rendimiento de la actividad docente. Las formas de hacerlo pueden ser, entre otras:

- *Dejar constancia de los objetivos que se buscan en cada momento.*
- *Utilizar un lenguaje claro, directo y conciso.*
- *Poner ejemplos reales y hacer comentarios que despierten su interés.*
- *Averiguar qué experiencias comunes pueden utilizarse como estímulos para el aprendizaje.*
- *Utilizar los medios y el material que se consideren estimulante.*

La ***orientación*** constituye también una labor fundamental. No hay que olvidar que, en último término, el factor decisivo en el aprendizaje es el trabajo personal de los alumnos. El docente da las pautas y después debe mantener hacia ellos una orientación que les permita trabajar solos, de acuerdo con su propio ritmo y al plan de trabajo que tienen trazado.

Además de estas tres actividades básicas, el profesor debe realizar una importante labor personal, como es la *preparación y actualización del material*, la *revisión de bibliografía*, la *puesta al día de sus conocimientos*, *la revisión de su programa*... Este continuo perfeccionamiento personal es fundamental en todas las áreas del saber, pero especialmente necesario en las Ingenierías, en los que la evolución es vertiginosa. En este sentido se considera de gran importancia que el docente realice también tareas de ***investigación***, puesto que ello tendrá influencia y una repercusión directa en su docencia.

La ***materialización*** de la actividad docente en las disciplinas universitarias se realiza a través de las clases (*teóricas o de problemas*), las prácticas y otras actividades complementarias, como los seminarios, tutorías... En los siguientes apartados se introducen brevemente cada una de las técnicas didácticas, haciendo hincapié sobre sus ventajas e inconvenientes, así como de la forma más eficaz de llevarlas a cabo.

3.5.2.1 Clases Teóricas o lección magistral

En la enseñanza universitaria, la lección magistral es la técnica de trabajo más antigua. De hecho, como método de enseñanza, nace con la misma Universidad, en la época medieval. Recogía la idea de *lectio* de las escuelas monacales, es decir, la lectura y comentario de un texto elegido como base de un curso. Actualmente, se ha convertido en la técnica más extendida y, lamentablemente, en muchos casos, la única, al existir limitaciones importantes, tales como *la masificación del alumnado, la dependencia del plan de estudios, medios...*

Su misión es la exposición completa, sistemática y ordenada del programa de la asignatura a lo largo del período lectivo de un curso académico.

Se trata de un tipo de enseñanza ocupada entera o principalmente por la exposición continua del docente. Aun cuando los estudiantes pueden preguntar o participar en una cierta discusión, su actividad fundamental es escuchar y tomar notas. La parte activa corresponde al profesor y presenta un carácter fundamentalmente instructivo. La ciencia se ofrece bajo la forma de una definición, solución o resultado, teniendo por tanto una enseñanza primordialmente temática.

Seguramente sea éste el método que más polémica despierta entre alumnos y profesores, existiendo tantos detractores, como defensores del mismo. Sin embargo, cuando aparecen los condicionantes anteriormente mencionados resulta difícil pensar en métodos más personalizados, o en los que se proponga un pleno contacto con el estudiante.

En los últimos años ha habido una fuerte tendencia opuesta a esta idea de clase magistral, y se ha defendido la denominada ***clase activa***. Lo que en principio es una idea válida, posibilitar que el alumno sea protagonista de su propio aprendizaje, se ha llevado en ocasiones a extremos exagerados. Además, hay un problema muy claro y que limita fuertemente la aplicación de esta idea: *el número de alumnos por clase*; lo que para un grupo pequeño puede ser correcto, pasa a ser totalmente inviable a medida que aumenta la cantidad de participantes. En realidad, clase magistral y clase activa, bien entendidas, deberían complementarse, y procurar, dentro de lo posible, una participación activa del alumnado, pero con la constante intervención del profesor, que siempre tendrá un papel importante que no podrá ser sustituido por los textos, ni por los modernos métodos audiovisuales o informáticos.

Inconvenientes de la lección magistral

A pesar de las críticas a las que se ha sometido, al descansar únicamente en la iniciativa del profesor, sigue siendo la pieza fundamental de la enseñanza, pues es el único momento en donde se hace una exposición coherente y completa de la materia, y donde el profesor tiene mayores posibilidades de influir sobre la comprensión de los conceptos por parte de los alumnos. Es por ello que en [Hale, 1964] se define la lección magistral

como "*un tiempo de enseñanza ocupado entera o principalmente en la exposición continua por parte del profesor*". Es en este punto, donde radican fundamentalmente los inconvenientes de la lección teórica; esto es, en la dificultad de conseguir una participación activa del alumno.

Se trata por tanto, de un método pasivo, debido a que el alumno se limita a tomar apuntes, y se preocupa básicamente de que éstos reflejen fielmente la explicación del profesor; lo cual supone un doble esfuerzo, pues posteriormente debe invertir gran cantidad de tiempo en asimilar los conceptos que se ha limitado a copiar en clase.

Esta pasividad, a la que se ve sometido normalmente el alumno, baja efectividad en la transmisión de los conocimientos, favorece la repetición, la omisión del sentido crítico, la rutina en la docencia y la ausencia de estímulo para el alumno, ya que generalmente son poco amenas. Al haber un único interlocutor, fluyen rápidas con pocas interrupciones, con lo que los estudiantes quedan abrumados por la cantidad de conocimientos que le son propuestos.

Por otra parte, se acentúa la idea de que el profesor es la única fuente del saber, se expone tan solo la visión personal del profesor sobre el tema, creándose una dependencia didáctica total. Como consecuencia, el alumno no adquiere el hábito de manejar bibliografía, ni desarrolla capacidad de síntesis ni de crítica.

Como inconveniente adicional, el profesor no tiene forma de conocer las características individuales de cada alumno, su formación previa. De esta forma se imparten los mismos conocimientos, al mismo ritmo y tiempo a todos los estudiantes por igual, obviando cualquier tipo de tratamiento personalizado o adaptados a las circunstancias concretas de cada estudiante.

Adicionalmente, esta falta de tratamiento personalizado redunda en la dificultad de controlar el proceso de aprendizaje de cada alumno. Esta ausencia de control, que permitiría comprobar de una forma continuada como se asimila el conocimiento, se manifiesta en que la única comprobación posible viene asociada normalmente a un examen, que difiere dicho control a un instante en el que el profesor no puede reaccionar.

Ventajas de la lección magistral

Hay que destacar, como principales ventajas, que es un buen método para introducir al estudiante en los conocimientos fundamentales de una materia. Ofrece al estudiante la posibilidad de disponer, sin demasiado esfuerzo por su parte, de información básica y actualizada sobre el tema, puesto que la labor de recopilación y estructuración recae sobre el profesor.

Así, las lecciones pueden presentar materia que no está aún en la bibliografía genérica de la asignatura. Por ello, se hace necesaria su aplicación en aquellas

disciplinas o partes de las mismas, en las que no existe apenas documentación al alcance del alumno.

Cuando por el contrario, la documentación sobre la materia es excesivamente abundante, el alumno en general agradece que, en la lección magistral, el profesor seleccione aquellos textos más recomendables.

Es, además, un método humano que puede ser de gran dinamismo dependiendo de las características del profesor. En este sentido, puede llegar a tener una fuerte capacidad motivadora en cuanto relaciona a unos profesores con una sólida vocación intelectual y a unos alumnos que se están iniciando en ella. Es, por tanto, fuente de una relación personal básica para una acción tutorial posterior.

Por otra parte, para la organización universitaria, es el medio más barato, pues no se necesita una gran dotación de recursos humanos y económicos. Esto se debe a que:

- Normalmente no es necesario más que la pizarra, en algunos casos acompañada por medios audiovisuales. Por tanto, no conlleva los costes de, por ejemplo, una práctica de laboratorio. Además, permite un cierto nivel de masificación que es impensable en otro tipo de técnicas, reduciendo el coste de recursos humanos.
- Permite una exposición más rápida de la materia por número de alumnos que cualquier otro método pedagógico.

Además, según **Fernando Lara** [Lara, 1997] la clase magistral a veces es más activa de lo que parece, pues el alumno al tomar apuntes traduce a su forma de pensar los contenidos que explica el profesor, si bien es cierto que esta afirmación, sólo se verifica mientras el alumno no haya perdido el hilo de las explicaciones.

Según los resultados de algunas encuestas [Beard, 1974], los estudiantes aceptan las clases magistrales, cuando son claras y constituyen resúmenes ordenados en los que se destacan los aspectos esenciales de cada tema.

En definitiva, la lección magistral puede ser perfectamente válida siempre que esté, por una parte, bien planteada y preparada, y por otra, se complete con actividades más personalizadas dentro de las limitaciones existentes.

Lección Magistral	
Ventajas	**Inconvenientes**
Exposición completa, sistemática y ordenada	Dificultad para conseguir la participación del alumno
El alumno dispone de información básica y actualizada	Acentúa la idea de que el profesor es la única fuente del saber
Puede llegar a tener una fuerte capacidad motivadora	No existe tratamiento personalizado del alumno
No es necesario una gran dotación de personal ni de medios	Dificultad de controlar en proceso de aprendizaje

Tabla 3.1. Ventajas e inconvenientes de la lección magistral

Aspectos influyentes en la calidad de la lección magistral

A pesar de que la lección magistral, como método pedagógico, se ve hoy en día cuestionada tanto por educadores como por estudiantes, desde una perspectiva realista la mayoría de las veces resulta difícil pensar en métodos de enseñanza alternativos allí donde los recursos humanos y materiales disponibles son escasos en relación con el número de alumnos. Por ello, hay que asumir y aceptar que éste seguirá siendo un método muy empleado en la enseñanza universitaria, de lo que surge la necesidad de conocer aquellos aspectos que contribuyen a su excelencia, o que por el contrario acentúan sus carencias.

Un aspecto importante de la lección magistral es su duración. Existen estudios que indican que la atención disminuye a partir de los ***40 minutos***, precipitándose de forma más intensa a partir de los ***60 minutos***, si bien, esto está sujeto a diferentes factores, como el horario, clases previas... Por ello, la duración de la lección teórica no debería exceder los **45-50 minutos**. Destacan en este sentido los estudios realizados por **D. H. Lloyd** [Lloyd, 1968] sobre los altibajos en el rendimiento del alumno y del profesor a lo largo de una lección magistral tal y como muestra la Figura 3.2.

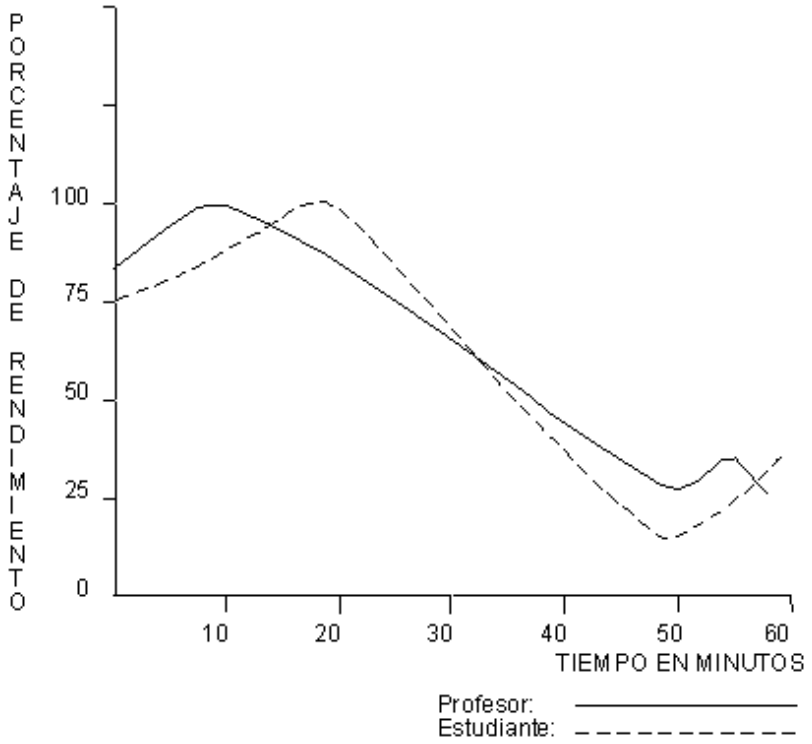

Figura 3.2. Rendimiento de Profesores y Alumnos a lo largo de una Lección Magistral

El relajamiento de la pendiente de estas curvas puede verse parcialmente favorecido por la utilización de técnicas audiovisuales que rompan la uniformidad de la oratoria del profesor. Generalmente los medios de transmisión del mensaje didáctico se dividen en:

- **Medios escritos**: recogen el conocimiento de una disciplina, y sirven para la formación del programa docente del profesor, así como de bibliografía de consulta del alumno.
- **Medios auditivos**: son los basados en la comunicación oral: charlas, clases magistrales, seminarios, unidades docentes en soportes de audio...
- **Medios audiovisuales**: son los relacionados con pizarras y medios proyectados, principalmente transparencias, pantallas de ordenador para retroproyector, cañón...

Según los expertos, el porcentaje de participación del sentido de la vista es del **83%** en el proceso de aprendizaje, frente al **11%** del oído. El número de datos retenidos en función del medio es mayor en situaciones en que se ve y se oye, que en el caso de ambos por separado, como se puede apreciar en los datos recogidos en la Tabla 3.2.

	Tiempo transcurrido	
Medio	***3 horas***	***3 días***
Oral	*70%*	*10%*
Visual	*72%*	*20%*
Audiovisual	*85%*	*65%*

Tabla 3.2. Duración del recuerdo de los contenidos de una exposición en función del tipo de medio utilizado

Parece conveniente, entonces, utilizar aquellos medios audiovisuales que permitan ilustrar la explicación y agilizar el desarrollo de la clase, sobre todo en temas con gran contenido de esquemas, diagramas y demás representaciones gráficas, que de otro modo pueden ocasionar confusiones, pérdidas de tiempo y concentración por parte de los alumnos y del profesor, al ser dibujadas manualmente sobre la pizarra.

Para ello, es adecuado facilitar con suficiente anterioridad el material utilizado, para que el alumno pueda disponer de él durante la clase. De esta forma, los estudiantes están en condiciones de seguir adecuadamente la explicación, sin que desvíen su atención en trasladar las indicaciones del profesor a sus apuntes, pudiendo completar la documentación aportada con breves anotaciones durante el desarrollo de las clases.

El poseer de antemano la documentación de la clase permite al alumno leerla con anterioridad (*utópicamente hablando*) con lo que el seguimiento de la clase se hace más llevadero. Pese a esta ventaja, el profesor ha de vigilar los peligros de suministrar documentación a los alumnos, que fundamentalmente son:

- Ausencia de interés por completar el material, y por tanto de manejar bibliografía y de aprender a cómo mantenerse al día por su cuenta en esa disciplina; pues el alumno tiende a asumir que el profesor en ningún caso va a exigirle desarrollar contenidos más allá de la documentación que este le aporta. Este punto se hace más notorio cuando la documentación consiste en los llamados "*apuntes del profesor*", que cuando por el contrario, se trata de

un conjunto de ilustraciones, esquemas, transparencias y referencias bibliográficas.

- Clases excesivamente veloces, pues el profesor al no verse frenado en sus explicaciones por la velocidad con la que los alumnos toman apuntes y/o copian el contenido de la pizarra, tiende a comprimir gran cantidad de materia en una sola clase. De esta forma se deja poco tiempo para la reflexión y, sobretodo, se fatiga al alumno, quien acaba por desconectar mucho antes del final de la clase. Por ello, una opción interesante en relación con la utilización de medios audiovisuales, consiste en basar el desarrollo de la clase utilizando la pizarra y usando transparencias como medio de apoyo, para presentar diagramas o esquemas.

Finalmente, la estructura de la lección magistral se constituye como un factor altamente influyente en su calidad. El seguimiento de una estructura correcta es un buen comienzo sobre el que construir la lección. Al inicio de la clase es conveniente dar una visión general del tema que permita seguir la exposición con facilidad. Al final de la clase, es aconsejable dar una conclusión que resuma lo expuesto. Si todavía no se ha llegado a dicha conclusión se debe de remarcar hacia donde se pretende llegar y mostrar el camino recorrido.

Aspectos que influyen en la calidad de la lección magistral
Duración de 45-50 minutos
Utilización de medios audiovisuales
Entregar al alumno de forma anticipada el material que se va a utilizar
El material entregado debe ser ilustraciones y esquemas que el alumno pueda completar durante la exposición
Estructura y exposición correcta que suscite el interés del alumno

Tabla 3.3. Aspectos influyentes en la calidad de la lección magistral

Entre el principio y el final debería de discurrir una exposición teórica correcta que suscite en el alumno el interés por el tema, motivándolo en el aprendizaje de la materia; avanzando con razonamientos claros, mostrando la relación entre los conceptos precedentes y/o consecutivos, y enfocada hacia los temas fundamentales, consiguiendo que el alumno no pierda en ningún momento una visión global de la asignatura.

La correcta exposición

La exposición se erige en el eje fundamental de la lección magistral, por este motivo se ha dedicado una sección aparte en la que tratar sus principales aspectos.

Para que los alumnos acepten, por tanto, la lección magistral, se hace necesario el correcto desarrollo de la exposición de la misma, para lo cual se deben tener en cuenta los siguientes aspectos:

- **Orden**: Salvo excepciones, la exposición debe seguir un esquema previamente establecido, aunque sin rigidez. En ningún caso se debe improvisar.
- Se debe intentar exponer los temas de forma **completa, sistemática** y **ordenada**. Cada tema forma una entidad completa pero no autosuficiente, y no puede aislarse de los demás. Por tanto, su exposición debe hacerse dentro del contexto global de la unidad docente a la que van destinados, y por consiguiente, de la asignatura. En este sentido, es necesario que el alumno disponga al principio del curso del temario completo que se va a desarrollar, para facilitarle el seguimiento de las clases, y que le sirvan de ayuda en su posterior estudio, de manera que pueda comprender de forma global la disciplina.
- **Motivación**: Se trata de atraer la atención del alumno, creando expectativas respecto a lo que se va a exponer (recursos: alusión a experiencias personales, suscitar problemas, cuestiones, aplicaciones, ejemplos oportunos, material audiovisual...).
- El profesor ha de **apoyarse en aquellos medios que sean apropiados al contenido** y ajustarse al mismo (no divagar). No se debe descuidar aspectos tales como al rigor, la precisión, la claridad y la amenidad en las explicaciones; evitando la monotonía, no sólo en el fondo, sino también en la forma, variando la entonación y ritmo de la exposición, (movimientos, gestos, entonación y ritmo adecuado; establecer pausas para la reflexión y resolver dudas).
- **Objetivos**: Es interesante poner en conocimiento de los alumnos los objetivos (a dónde se quiere llegar), su relación con otras materias, límites del conocimiento sobre el tema... Como ya se ha comentado, el inicio y final de la clase es un momento ideal para remarcar los objetivos. No obstante puede ser conveniente refrescar a lo largo de la exposición la perspectiva real del desarrollo de la lección.
- **Resumen**: es importante resumir de vez en cuando los aspectos importantes, recapitular a lo largo de la exposición y, sobre todo, realizar una síntesis que indique al alumno qué es lo más importante del tema tratado. Los resúmenes no sólo han de hacerse al final de cada lección/clase, ya que el resumen le sirve al profesor de recapitulación para recalcar lo que considere importante en cada momento, y aclarar lo que haya detectado que no está entendido sobradamente.
- **Prácticas**: es también importante, siempre que sea posible, ejemplificar la teoría y presentar situaciones prácticas. Evidentemente, se optimiza el método, si se realizan prácticas fuera de clase sobre el tema.

- **Capacidad de Comunicación**: Mantener una actitud abierta y relajada para conseguir una comunicación con los estudiantes, de forma que les incite a plantear preguntas en clase; invitando al alumno a exponer las dudas que le surjan durante la clase, siempre y cuando no se perjudique el ritmo de la exposición.
- **Observación y Comprensión hacia el Esfuerzo y Capacidad del Alumno**: El profesor debe ser capaz de en todo momento de sondear el seguimiento de sus explicaciones por parte de la mayoría de los alumnos, insistiendo sobre los conceptos fundamentales hasta que queden suficientemente claros. Es preferible incumplir la totalidad de los contenidos finales que se habían puesto como meta en el programa, que por cegarse en mantenerlos, forzar la marcha, huyendo hacia adelante, pensando que "*cuando los alumnos se lo estudien, ya lo entenderán*". Por el contrario, el hecho de que unas explicaciones están construidas a partir de otras anteriores, provoca que la falta de compresión de los conceptos ya explicados haga inútil el esfuerzo del profesor por introducir los nuevos.

Se debe de evitar por norma, que la explicación de los conceptos importantes coincida con el final de la clase, ya que esto conlleva dos peligros:

- Dejar la explicación de dichos conceptos inacabada.
- Acabar la explicación, pero haciéndola coincidir con el momento en que los alumnos están menos receptivos. Esta falta de receptividad se ve motivada fundamentalmente por el cansancio y, en el caso de que los conceptos a explicar estén basados en otros introducidos en la misma clase, en la falta de un período de reflexión/maduración que permita asimilar los nuevos conceptos a partir de los precedentes.

3.5.2.2 Clases de problemas

Un aspecto importante dentro de la enseñanza son las clases de problemas. En ellas, el alumno debe aplicar los conocimientos teóricos adquiridos para resolver problemas o supuestos prácticos, constituyendo un eficaz factor de realimentación de los mismos. Su adecuada inclusión dentro del desarrollo de la materia permitirá reforzar y aplicar los conceptos expuestos en teoría, y fomentar en el alumno la capacidad de análisis y síntesis.

Por tanto, no parece conveniente establecer una división tajante entre clases teóricas y de problemas, sino entremezclar ambas, de modo que las exposiciones teóricas se alternen con ejercicios ilustrativos o aclaratorios. Asimismo, el profesor puede obtener una información muy valiosa de estas clases, ya que le permiten detectar dificultades de comprensión y aplicación de los conceptos teóricos.

Parece interesante proponer, para afianzar los conocimientos teóricos, un conjunto de problemas, junto con alguna sugerencia de cara a su solución, que deben ser resueltos por los alumnos mediante su trabajo personal. De esta forma el alumno completará el proceso de aprendizaje, razonando y enfrentándose a dudas y a conceptos poco claros, que de otro modo no se habrían presentado. Posteriormente, el profesor puede averiguar cuál es el grado de éxito en su solución, y orientar y resolver los más complejos con los alumnos. En caso de dificultades especiales, es aconsejable organizar un seminario para tratar estos problemas.

Los enunciados de los problemas, así como las soluciones aportadas por los alumnos y corregidas por el profesor pueden dejarse disponibles en Internet, si éste último lo estima conveniente.

3.5.2.3 Clases de prácticas

Las prácticas constituyen uno de los complementos ideales e imprescindibles de las clases teóricas y de problemas en la educación universitaria. Esto es más acusado en las disciplinas relacionadas con la Informática.

La realización de las clases prácticas sirve al alumno para adaptar los conceptos teóricos estudiados en clase, en el entorno real de aplicación de éstos, sobre casos concretos. El profesor debe tomar parte activa en la clarificación de las posibles dudas o controversias que surjan, pues es en ese momento cuando el estudiante está más receptivo. Asimismo, debe participar como observador atento a las respuestas de los educandos, con objeto de conseguir evaluar sobre la marcha la adecuación del programa docente y de la técnica de enseñanza utilizada. Para mejorar el rendimiento es conveniente que el alumno disponga previamente de los supuestos prácticos que debe analizar y desarrollar.

Respecto del desarrollo de las prácticas, se debe indicar primero que, dado el elevado número de alumnos es inevitable dividir al alumnado en grupos de prácticas. En concreto, al tener que desarrollar esta docencia a través de prácticas con ordenador, se tiene que fijar un número máximo de alumnos por ordenador, dato este que, junto con la capacidad del aula, permitan obtener el número de alumnos por grupo.

Se puede pensar que el trabajo de dos y tres personas por ordenador es negativo frente el trabajo individual. Sin embargo, no es menos cierto que este hecho facilita la distensión de la dinámica de la clase, haciendo posible una relación fluida entre el profesor y los alumnos, y entre los propios alumnos, permitiendo la discusión dentro del grupo y el seguimiento de la labor de cada alumno por parte del profesor. Por otra parte, es muy probable que su futura actividad laboral se desarrolle dentro un equipo formado por varios profesionales.

Existen dos planteamientos distintos a la hora de desarrollar las prácticas: *Prácticas Guiadas* y *Prácticas Libres*. En ambos tipos, la experiencia dice que los estudiantes

acogen las prácticas con un elevado interés. De cualquier modo, se debe contar, por supuesto, con los medios necesarios para llevarlo a cabo.

Prácticas guiadas

Son aquellas que son dirigidas directamente y en todo momento por el profesor. Se corresponden a prácticas en las que se introducen entornos de trabajo o utilidades que son nuevas para el alumno.

En este tipo de prácticas es conveniente hacer notar a los alumnos la conveniencia de *parar* al profesor en todo momento en que sean incapaces de ejecutar una determinada acción con éxito en el ordenador; pues de lo contrario, se arriesgan a quedar desconectados del seguimiento de la clase.

Por este motivo en este tipo de prácticas es contraproducente aumentar el número de terminales por grupo, ya que el profesor tiene que hacer un seguimiento estrecho del avance paso a paso de los alumnos, haciéndose más probable que los alumnos de un determinado puesto se "*atasquen*", cuanto más puestos haya. Cada vez que los alumnos de un puesto tienen un problema en este tipo de prácticas, requieren normalmente la atención exclusiva del profesor para resolverlo; provocando que el resto de alumnos se queden esperando a que el profesor pueda continuar guiando el resto de la práctica.

Prácticas libres

Otro tipo de prácticas son aquellas en las que el profesor presentará brevemente la práctica, indicando sus objetivos y cómo debe progresarse en la misma, de modo que el alumno, ayudado del guión que se le ha proporcionado con anterioridad, la lleve a cabo. El profesor debe actuar como director, moderador y observador, atento ante las dudas o problemas que surjan, cuidando que los alumnos progresen de forma simultánea.

En cuanto a la organización temporal de las sesiones, será tal que las prácticas siempre sean posteriores a la explicación teórica de los conceptos, y de forma que se puedan realizar íntegramente con una continuidad temporal en temas afines que las haga más provechosas.

Esto tampoco quiere decir que sea conveniente el introducir absolutamente todos los conceptos en las clases teóricas previas a las prácticas, pues de lo contrario las prácticas se convierten en una mera verificación por parte del alumno de lo que se le ha enseñado en la teoría, perdiendo así dinamismo y participación que surge cuando intencionadamente el profesor no presenta todos los conocimientos necesarios para resolver dicha práctica.

El profesor de esta forma queda a la espera de que el alumno descubra por si solo la dificultad de un determinado problema, a sabiendas de que los alumnos, normalmente, fracasarán en el intento de realizarlo por ellos mismos. De esta forma se tiene que, por un lado el alumno descubre cómo no ha de resolverse el problema, y por otro se fuerza a

que acabe solicitando la ayuda del profesor para poder seguir trabajando. Es en este momento cuando el profesor debe de aprovechar para explicar los conceptos que vengan al caso y que intencionadamente se dejaron sin matizar en la exposición teórica, pues el alumno ha reflexionado sobre el problema y, además, suele estar más receptivo de lo habitual.

En algunos casos puede ser conveniente hacer presentar al alumno un informe detallado con los resultados y las conclusiones obtenidas de cada una de las prácticas. Este informe puede ser presentado de forma individual, a pesar de que las prácticas se hayan realizado en grupo. De este modo, cada estudiante se ejercita en la ordenación sistemática de los conocimientos y aprende a estructurar un trabajo.

Los enunciados de las prácticas, así como los informes de las prácticas corregidas pueden dejarse disponibles en Internet si el profesor lo estima conveniente.

3.5.2.4 Actividades docentes complementarias

Las actividades complementarias contribuyen a mantener el interés del alumno y a favorecer un contacto más directo con el profesor y con el mundo profesional, propiciando un sistema de educación adicional muy útil para su formación.

Seminarios y conferencias

Existente temas que no pueden ser tratados en toda su extensión durante un curso académico, bien por la propia limitación del tiempo asignado a la asignatura, bien porque un adecuado tratamiento de los mismos requiere del concurso de expertos, o bien por la inexistencia de instrumentos o medios especiales. En estos casos parece oportuno la utilización de seminarios o cursos monográficos.

Los seminarios y conferencias son reuniones organizadas con el propósito de incrementar el conocimiento general, completando la formación del alumno en el campo de los conocimientos y de la práctica profesional. Estas actividades son útiles para el fomento de la participación del alumno en la educación.

Asimismo, permiten la exposición de temas con un enfoque diferente al de la clase habitual. En ellos pueden presentarse aspectos muy específicos, avanzados o de interés general, y también una visión clara del mundo laboral.

Los seminarios, en concreto, se pueden plantear desde varios enfoques. En primer lugar, el profesor puede exponer un tema específico y, posteriormente, debatirlo con los alumnos en un ambiente más distendido e informal que el de las clases, intentando que los alumnos expongan sus dudas, comentarios o sugerencias. La discusión puede estar precedida de una conferencia o simplemente de algunas observaciones del individuo que representa el papel de director. Éste con antelación habrá preparado un esquema general para dirigir la discusión hacia determinadas metas.

Otra opción consiste en que sean los propios alumnos los que presenten un tema, previamente preparado, para proceder a continuación a su debate y discusión con el resto de los alumnos y el profesor. Este tipo de actividades sirve para fomentar las facultades expositivas de los alumnos y promover la crítica y la creatividad.

Operando de esta forma, se logran los objetivos que se deben confiar al seminario, que principalmente son:

- **Crear el hábito de investigación científica**: inculcar el espíritu científico, desarrollar en los alumnos la técnica del pensamiento crítico y del pensamiento original. No obstante hay que ser cuidadosos en este punto y no olvidar que, en el caso que se discute, se está ante alumnos de primer ciclo, en este sentido el profesor debe de intentar guiar los trabajos hacia aspectos tan concretos como prácticos y evitar que el alumno derive hacia temas demasiado teóricos y/o abstractos que desborden su capacidad actual de análisis. En este mismo sentido, se observa que las preguntas de los alumnos durante el seminario disminuyen drásticamente cuanto menor sea el cariz práctico de los mismos.
- **Aprendizaje de los métodos científicos**: se trata de enseñar a manejar los instrumentos del trabajo intelectual; entre ellos destacan el manejo de bibliografía y la experimentación con los medios existentes.
- **Mejorar las capacidades de expresión escrita y oral**: el alumno tiene que elaborar trabajos escritos y defender sus puntos de vista. El profesor velará:
    1. Por que los trabajos no sean recortes de las fuentes bibliográficas consultadas, y que detrás de ellos exista una auténtica labor de síntesis y de crítica. Los trabajos escritos que sirvan de base a los seminarios, pueden igualmente ser publicados en la página web de la asignatura para que el esfuerzo de los alumnos sea accesible a las promociones venideras.
    2. Por que el alumno exponga con brevedad y prescindiendo de los elementos que sean innecesarios para comprender la conclusión final de su trabajo.

El papel del profesor en los seminarios se identifica más con el de coordinador y moderador que con el de estrictamente docente. Así, el profesor con su presencia, favorecerá la creación de un clima de confianza, siendo el encargado de mantener la discusión dentro de sus límites, minimizando el debate en asuntos que no tengan importancia, relacionando una discusión con las anteriores y animando a los alumnos e invitados a participar, procurando que la discusión no sea monopolizada por unos pocos. El profesor resumirá y cerrará cada tema discutido.

De cualquier modo, siempre que se realicen este tipo de trabajos, se debe contar, por supuesto, con los medios necesarios para llevarlo a cabo. Es necesario disponer de una biblioteca bien dotada, acceso a Internet y del equipo especializado en el aula de informática.

Por otra parte, se pueden organizar conferencias con la colaboración de profesores de la propia titulación, o de otras titulaciones, o incluso de otras universidades, y otras personas del mundo profesional. Éstos últimos suscitan un interés especial, ya que los conferenciantes conocen de cerca las exigencias y peculiaridades del mundo laboral, poniendo en sus enseñanzas un vigor y autenticidad que es muy apreciada por los estudiantes. Esto permite al alumno conocer puntos de vista diferentes, problemas y últimos adelantos, que le aproximen al entorno en que realizará su ejercicio profesional.

En general, los estudiantes acogen estas actividades docentes con un marcado interés, si bien:

1. Para lograr una planificación correcta de los seminarios y conferencias, hay que tener en cuenta la disponibilidad de tiempo de los alumnos, de acuerdo con sus horarios de clase y otras actividades docentes. Hay que tener presente, por ejemplo, que si bien pueden organizarse seminarios y conferencias al principio de curso, pues se dispone de más tiempo para ello, el nivel de conocimiento en esta época es escaso. No obstante, al principio pueden plantearse aspectos de interés general, para organizarse paulatinamente sesiones sobre aspectos más específicos.
2. Inicialmente los alumnos prefieren este tipo de método frente a los tradicionales. Sin embargo, a medida que se pone en práctica se quejan de la cantidad de tiempo que les puede llegar a absorber. Por tanto, se hace necesario que el profesor calibre en cada caso el nivel de profundidad esperado de los trabajos.

Visitas y prácticas en instalaciones y centros profesionales

En general sería conveniente llevar a los estudiantes a los centros donde desempeñarán sus labores profesionales. Es interesante que los estudiantes vean de cerca los centros donde desarrollan su trabajo los especialistas en la materia, a fin de conocer de cerca la realidad de las enseñanzas explicadas en la Universidad, y para que adquieran una visión directa y global del funcionamientos de esos centros. Otro beneficio a obtener de este tipo de contactos es el que el empresario conozca la formación de los alumnos con vistas a una posible contratación futura.

Sin embargo, la carga docente que tienen los alumnos hace muy difícil coordinar diferentes visitas a centros profesionales. Además, algunos estudiantes ven estas salidas como jornadas festivas de las que sacan muy poco provecho.

Adicionalmente, el plan de estudios de Ingeniería Técnica en Informática de Sistemas articula otros medios para que los alumnos tomen el pulso a la actividad laboral de forma compaginada con sus estudios:

1. Se pueden conceder **6 créditos** de libre disposición por un mínimo de **180 horas de trabajo**, debidamente justificadas, en empresas u organismos públicos, en tareas de programador o equivalente.
2. Por otro lado los proyectos de fin de carrera *(9 créditos obligatorios en el tercer curso*) pueden desarrollarse en ocasiones en el marco de la colaboración **Universidad–Empresa**, por lo que también pueden servir a este fin.

3.5.2.5 Tutorías

El sistema de tutorías es un aspecto importante en el proceso de enseñanza, pues permite que el alumno pueda contar con la posibilidad de realizar consultas, discutir y esclarecer dificultades surgidas en las clases u otras actividades docentes, fuera del horario normal de éstas. El Real Decreto 898/1985 de 30 de abril, sobre régimen de profesorado universitario, fija en ***seis a la semana*** el número de horas de tutoría para el Cuerpo de Profesores Titulares de Escuela Universitaria.

Representa esta técnica otra de las que tradicionalmente se han señalado para impartir la enseñanza universitaria, si bien su aplicación más fiel se reduce principalmente a la Universidad Anglosajona. El método consiste en una reunión periódica del estudiante (solo o en pequeños grupos), con el profesor tutor. En esta reunión, la discusión por medio del diálogo permite el intercambio de ideas, como base para la orientación y el desarrollo de la capacidad del alumno. Se diferencia del seminario, aparte de por el número de participantes, en que no existe excesiva rigidez en el tema y se concede al estudiante más iniciativa y responsabilidad. En definitiva, el alumno aprende por tres vías sucesivas: al hacer el trabajo, al observar los errores cometidos y defender los puntos que considera acertados y, finalmente, al repasar el trabajo completo, corregido y comprobándolo con su primera versión.

En la Universidad española, dado el desequilibrio existente entre el número de alumnos y profesores, es razonable considerar el sistema como impracticable. Ahora bien, las posibilidades que presenta este método, como complemento de los anteriores, hace considerar como deseables la aplicación del mismo en formas menos estrictas, pero igualmente válidas y tendentes a la, hasta hoy utópica, "*enseñanza individualizada*".

Las tutorías sirven para poner en contacto directo al profesor con los alumnos, fomentando la relación entre ambos. Este contacto mutuo debe ser utilizado por ambas partes. Los alumnos para consultar al profesor todas las dudas surgidas, inquietudes, opiniones, perspectivas profesionales, buscar bibliografía específica... El profesor debe

buscar en dicho contacto los elementos de autoevaluación que le permitan detectar el grado de entendimiento y las dificultades que encuentran los alumnos en la materia, detectando los conceptos captados de forma deficiente y que, por tanto, necesiten una revisión del planteamiento expuesto en las clases teóricas o prácticas. En definitiva, el profesor puede y debe observar en las tutorías la marcha general y particular de la asignatura.

La experiencia demuestra que, a pesar de que pueda ser beneficioso para la enseñanza de los alumnos, éstos son reacios a la hora de utilizar las tutorías, aunque las dudas existan. Desgraciadamente, las consultas se reducen a los días previos del examen, que es cuando el estudiante concentra mayores esfuerzos en la asignatura.

Es por ello que el profesor debe de fomentarlas, buscando la actitud participativa de los alumnos, concienciándoles de que el profesor está a su servicio. La preparación de trabajos y seminarios por parte de los alumnos constituyen la forma más eficaz de forzar al alumno a utilizar las tutorías. Si las fechas de entrega de estos trabajos se dosifican convenientemente a lo largo del curso, hasta el alumno más reacio, acaba por pasarse varias veces por el despacho para solventar las dudas que surgen en la elaboración de dichos trabajos. De esta forma se fomenta el trabajo diario de aprendizaje, frente a la extendida actitud de estudiar las materias por bloques en vísperas de los exámenes, que al fin y al cabo constituye la causa última de que las tutorías no se utilicen, o sólo se utilicen en estas fechas.

Conviene mentalizar a los alumnos de que, para la utilización de este servicio, hay un horario establecido que deben conocer y respetar para que tanto profesores como alumnos puedan programarse sus quehaceres. Una iniciativa en este sentido, consiste en pactar con los alumnos cuáles van a ser las horas de tutoría. Éstas en ningún caso deberían de coincidir con sus horas de clase; y en el caso de que el profesor impartiera clase a grupos con diferentes horarios debería de hacer lo posible por distribuir sus horas de tutoría de forma que cualquier alumno pueda acceder a las mismas sin tener que ausentarse de clase.

No obstante, no ha de tomarse el horario de tutorías como limitación en la iniciativa de los alumnos, por lo que, siempre que sea posible, el profesor debe estar dispuesto para atender cualquier consulta.

3.5.2.6 Internet como vía de comunicación con los alumnos

En el IS'97[7] [Davis et al., 1997], se recomienda la utilización de Internet en la actividad docente. Esta recomendación en principio es muy vaga, pero se ve refrendada por la facilidad que actualmente tienen los alumnos para acceder a Internet, bien desde las propias Escuela y Facultades, bien desde sus propios domicilios.

[7] El IS'97 es el curriculum de ACM para Sistemas de Información (Information Systems 97).

La aplicación de los servicios que ofrece a Internet a la docencia universitaria son enormes, destacando el correo electrónico y de una manera especial la WWW [Vetter and Severance, 1997], [García Carrasco et al., 1999].

El correo electrónico ofrece un medio de comunicación asíncrona entre el alumno y el profesor [Nishida et al., 1996], que permite suprimir algunos de los inconvenientes de las tutorías presenciales, especialmente en lo referente a incompatibilidades de horarios, o distancia geográfica entre el alumno y la Universidad, o en la Universidad a Distancia [García Carrasco et al., 1999]. También permite el intercambio de ficheros entre las dos partes, *memorias e informes de prácticas* por parte de los alumnos, *material bibliográfico* por parte profesor, por ejemplo en los proyectos final de carrera.

La web ofrece muchas posibilidades: la mera exposición de los programas docentes, revistas en línea [Riser and Gotterbarn, 1998], acceso a almacenes de información (referencias bibliográficas, artículos, informes...) que juegan el rol de bibliotecas digitales [Marchionini and Maurer, 1995], soporte de la información relacionada con un curso, una asignatura o un libro (ya sea de forma pública para todo el mundo, o restringida a una Intranet) [Veraart and Wright, 1996], [Paxton, 1996], preparación de cursos de forma conjunta, impartición de cursos de forma no presencial...

El material de carácter docente recopilado y publicado en páginas web es un activo de gran valor para la preparación de cursos o asignaturas [Mercuri, 1998].

Un interesante ejemplo de la utilización avanzada de la red para la docencia es, en el terreno de las bases de datos, el caso de **Jeffrey Ullman** y **Jennifer Widom** en la universidad de Standford[8], en la que, como soporte a su libro [Ullman and Widom, 1997], se accede a resúmenes sobre manuales de productos utilizados en las prácticas por los alumnos de esta universidad, transparencias utilizadas en las exposiciones de las clases y soluciones de exámenes.

La utilización de la potencia que ofrece el servicio web se está explotando en las asignaturas propias del perfil de este Proyecto Docente[9]. Donde, además del programa de cada asignatura, se proponen las prácticas (enlazándose sus soluciones), se facilita material complementario (en forma de artículos complementarios, manuales, informes técnicos, problemas resueltos...) y se establece un canal donde anunciar asuntos relacionados con la asignatura (noticias, cambios de fechas, defensas de prácticas...). Además, la organización de este material por cursos académicos ofrece un histórico muy interesante a los alumnos de las siguientes promociones, en forma de ejercicios resueltos y referencias de consulta.

En general, este sistema presenta las siguientes ventajas:

[8] http://www-db.stanford.edu/~ullman/fcdb.html

[9] http://tejo.usal.es/~fgarcia/docencia.html

- Permite la confección de material adicional de la asignatura de una forma en la que el alumno está plenamente involucrado. Todo este material que se elabora, tiene el denominador común de no estar en principio al alcance del alumno. En concreto se trata de libros de problemas solucionados, tutoriales sobre herramientas y entornos, artículos o informes sobre temas específicos...
- Desde el punto de vista de los alumnos que hacen los trabajos se favorece la relación profesor-alumno, potenciando el uso de las tutorías más allá del día antes del examen.
- En el caso de los trabajos de alumnos corregidos por el profesor, aquellos alumnos que posteriormente accedan a los mismos, tendrán la opción de, por un lado, ver cómo se resuelven correctamente los ejercicios, y por otro, observar los fallos más comunes en los que se puede incurrir al intentar solucionarlos, obteniendo una experiencia positiva de los fallos de los compañeros.

La desventaja fundamental es la cantidad de tiempo de tutela y corrección de trabajos a llevar a cabo por el profesor, por lo que sólo se puede aplicar a grupos no numerosos (optativas y grupos de voluntarios en las obligatorias y troncales).

En cierto modo, este método se asemeja a los seminarios en cuanto que también son desarrollados por los alumnos, pero se diferencian fundamentalmente en que la experiencia de los alumnos queda *congelada* en la red a disposición de sus compañeros actuales y de las promociones venideras. Además, no es tan importante la corrección del resultado final del trabajo, como el obtener los fallos más comunes de los alumnos. Este último dato no sólo es interesante para el alumno, sino que permite a la larga dar una idea precisa al profesor de qué puntos no acostumbran a quedar claros, y dónde debe de hacer por tanto mayor hincapié.

Otra posibilidad que ofrece la red es dejar en ella manuales, tutoriales y documentación diversa elaborada por los fabricantes, sobre los productos que se utilizan en las prácticas de las asignaturas. En este sentido conviene asegurar que aquellos elementos que no sean de libre disposición sólo sean accesibles en Intranet, al no ser legal exponerlos al público en general. Sin embargo, si es legal y necesario que sean accedidos por los alumnos de la titulación.

Como nada es perfecto, el abuso o la delegación excesiva de Internet como base a la docencia también tiene sus peligros en forma de falta de seguridad, deshumanización de la docencia, problemas de derechos de autor... [Neumann, 1998], [Mercuri, 1998].

3.6 Sistemas de evaluación

La evaluación es un proceso inseparable del ciclo enseñanza – aprendizaje, que tiene como fin el determinar en qué medida se han logrado los objetivos educativos establecidos sobre la base de las tareas profesionales concretas que el estudiante debe ser capaz de realizar al finalizar su aprendizaje. Constituye un aspecto tan esencial como complejo y delicado en el proceso docente.

La evaluación no se reduce a una medida de la asimilación de la materia a lo largo del curso, sino que debe aportar los datos necesarios para valorar los programas, los métodos didácticos... Así, con respecto al profesor, la evaluación actúa como una realimentación directa destinada a mejorar el proceso educativo.

A través del proceso de evaluación se pretende alcanzar los siguientes objetivos:

a) Poder llegar a tener un juicio de valor sobre la realidad del rendimiento del alumno, así como de sus posibilidades reales. Esta información es sumamente útil de cara a medir el grado de aprovechamiento del alumno en el marco de una enseñanza personalizada.

b) Puesta de manifiesto de la coherencia o no entre los objetivos y los resultados. Si en este análisis la distancia entre lo deseable y lo real es aceptable, no habrá que introducir ninguna corrección, si, en cambio, las desviaciones no son las deseadas, habrá que tomar las decisiones pertinentes para corregir las deficiencias en los elementos que las hayan provocado.

c) Recoger información para la toma de decisiones, no sólo con respecto al funcionamiento del sistema, sino, especialmente, en relación con los elementos que lo componen.

Según **A. de la Orden** [Orden, 1985], las cuestiones básicas que afectan a la evaluación educativa se centran en los siguientes puntos:

1. Determinación de lo que se ha de evaluar y de los procedimientos y formas de evaluación.
2. Establecimiento y formulación de los criterios de evaluación.
3. Determinación y clarificación de las decisiones que han de ser adoptadas como resultado de la evaluación.

Para que un sistema de evaluación sea válido, debe de reunir las siguientes condiciones:

a) Que el alumno conozca con precisión los objetivos formulados.

b) Que las pruebas exijan una adecuada y representativa muestra de los contenidos y conductas especificadas en los objetivos.

c) *Que la evaluación sea fiable y objetiva, de tal forma que el azar o los errores instrumentales tengan un efecto mínimo en los resultados.*

La evaluación debe realizarse con un rigor especial, esto es, debe ser lo más objetiva posible, tratando por igual a todos los estudiantes. Además, debe estimarse más la capacidad de aplicación de los conocimientos, que su mera memorización. Así, debe procurar evitarse que las cuestiones planteadas en las pruebas tengan un carácter de "*recuerdo aislado*" o memorización mecánica, a la vez que deben incluirse aquéllas que sirvan para detectar la capacidad de aplicación, síntesis y generalización de conceptos; evitando así lo que **Fernando Lara** denomina "*un aprendizaje puramente memorístico que se perderá en cuanto el alumno haga el examen, o tras la caña de después*" [Lara, 1997].

En general, dentro de un proceso de evaluación se admite la necesidad de pruebas iniciales y finales.

Las **pruebas iniciales** tienen carácter de evaluación formativa, no de certificación. Entre ellas se distinguen dos tipos de pruebas diferentes en cuanto a su finalidad, pero que pueden realizarse de forma simultánea. Estas son:

- ***Prueba del nivel requerido***: Antes de comenzar una enseñanza dada, debe asegurarse que los estudiantes estén en un cierto nivel: *nivel requerido*. Corresponde al profesor definir cuáles son los conocimientos que juzga indispensables para que los estudiantes que le son confiados, puedan beneficiarse al máximo de la enseñanza que se les dará. Esta prueba permite conocer si todos los estudiantes están al mismo nivel y si este coincide con el requerido y en su defecto efectuar la modificación necesaria de la enseñanza prevista para colocarles a nivel. Si esto no se hace, la calidad disminuye necesariamente. Esta cobertura puede realizarse de diferentes modos: referencias bibliográficas, clases suplementarias a los estudiantes afectados y sólo en último caso, clases previas para todos.
- ***Prueba de nivel de partida***: Su finalidad es conocer, al comienzo de la enseñanza, el nivel al que están los estudiantes respecto al contenido en sí de la asignatura. Solamente conociendo el nivel de punto de partida es posible calcular la ganancia real al final de la enseñanza. Por otra parte, si se detectara que algunos estudiantes estuvieran bastante avanzados respecto a los objetivos previstos, convendría considerar una orientación suplementaria para los mismos; situación que por otra parte no es frecuente.

Las **pruebas finales** siempre tienen carácter de certificación. Si un estudiante juzgado competente obtiene una mala calificación en el examen final, debe reevaluarse la situación y no dejar al examen final una exclusiva misión sancionadora.

La evaluación debe ser un proceso continuo y como tal no puede limitarse a una prueba aislada, con sentido de sanción arbitraria sin tener en cuenta más factores. La

denominada *evaluación continua* se convierte así en el método de evaluación más adecuado, ya que tendría en cuenta todos los aspectos de la labor del estudiante (*participación en seminarios, trabajos realizados, nivel de participación, resultados en exámenes parciales y/o finales*...).

Una evaluación continuada supone la mejor forma de conocer la evolución de los conocimientos del estudiante, así como de constatar el grado de asimilación alcanzado. En la mayoría de los casos se demuestra que este sistema es inviable dentro de la estructura universitaria, debido principalmente a dos aspectos. Por un lado, el gran número de alumnos obliga al profesor a generar, corregir y supervisar una gran cantidad de actividades docentes. Por otro, el alumno, al menos inicialmente, se opone a realizar, preparar y asistir a esas actividades, debido generalmente, a que suponen un gran esfuerzo por su parte.

3.6.1 Técnicas de evaluación

Si se considera que el desarrollo de una tarea profesional comporta una serie de habilidades, que al dirigirse a otra persona implican una cierta capacidad de comunicación, y que para su cumplimiento es necesaria determinada capacidad intelectual y un cierto nivel de conocimientos. Cualquier metodología para la evaluación de estudiantes debe disponer de técnicas destinadas a la evaluación de estos tres campos:

1. *Conocimientos o proceso intelectual.*
2. *Capacidad de comunicación.*
3. *Habilidades prácticas.*

Evidentemente el tipo de instrumentos que proporcionan los datos para emitir juicios de valor sobre cada uno de estos campos es distinto. Ninguna prueba por muy fiable y objetiva que sea puede utilizarse de forma exclusiva. El uso de una mayor variedad de técnicas asegura una evaluación más completa, mientras que el limitarse a una de las pruebas tradicionales orales o escritas, sólo permite determinar la capacidad de memorización o nivel de conocimientos teóricos, pero no la aptitud para la utilización profesional de esos conocimientos.

En la Tabla 3.4 se recogen las técnicas que preferentemente se emplean para la evaluación de los estudiantes en los tres campos referidos.

Seguidamente se procede a la exposición de las características de cada una de ellas, así como a la valoración de sus ventajas e inconvenientes, indicando algunas pautas para su correcta formulación.

	Métodos directos	Métodos indirectos

Campo de los conocimientos	No existen	• Pruebas escritas
		  ○ Preguntas de Respuesta Libre (Redacción)
		  ○ Preguntas de Respuesta Abierta Corta (PRAC)
		  ○ Preguntas de Elección Múltiple (PEM)
		  ○ Situación Problemática
		• Pruebas orales
		  ○ Exposición
		  ○ Debate
		  ○ Entrevista
		• Informes de Observación

Campo de la comunicación y de las habilidades profesionales	Observación de una prueba práctica: • Situación real • Entrevista	Proyecto

Tabla 3.4. Técnicas de evaluación más utilizadas

3.6.1.1 Evaluación del campo de los conocimientos

Como puede deducirse de la tabla anterior, el examen, independientemente de la forma que adquiera, se convierte en la forma más utilizada para evaluar los conocimientos del alumno en relación con una determinada materia.

El examen debe poseer la mayor capacidad evaluadora posible, además de servir de realimentación eficaz entre docentes y discentes, para así valorar correctamente la asimilación de las materias tratadas [Crawford and Fekete, 1997]. El examen debe ser planteado de forma que disminuya la incertidumbre derivadas de la prueba en sí, siendo deseable su compleción con respecto al temario, su eficacia en relación con los objetivos planteados y que el alumno se encuentre informado sobre los conceptos que se le exigen y la forma cómo se va a comprobar este conocimiento [Mora et al., 1995].

En general, se pueden citar una serie de criterios a tener en consideración, en cuanto a exámenes se refiere [Gullbert, 1989]:

- **Validez:** grado de precisión con que el examen utilizado mide verdaderamente aquello para lo que fue diseñado como instrumento de medida.
- **Fiabilidad:** como grado de confianza que podemos adscribir a los resultados de un examen. Se trata de la constancia con la que un examen nos aporta los resultados esperados.
- **Equilibrio:** grado de concordancia entre la proporción de preguntas reservadas a cada uno de los objetivos y su proporción ideal del examen.

- **Equidad:** grado de concordancia entre las preguntas propuestas en el examen y el contenido de las enseñanzas.
- **Discriminación:** cualidad de cada elemento de un método de evaluación, que permite distinguir, con respecto a una variable dada, a los alumnos más preparados de los menos preparados.
- **Tiempo:** es bien conocido que un instrumento de medida será menos fiable si, a causa de disponer de poco tiempo, permite introducir factores no pertinentes como el azar. De todas formas, depende del objetivo que se persiga, a veces el fijar el tiempo estrictamente es un factor importante.

En pro de garantizar la objetividad de los exámenes, parece esencial cuidar al máximo su corrección, la cual debe de ser repasada convenientemente. Asimismo, deben ponerse los medios oportunos para que las revisiones de los exámenes sean conocidas y accesibles por todos los alumnos, haciendo de las mismas una extensión de las tutorías. Este punto permite que el alumno tome conciencia de su responsabilidad directa en la nota obtenida, además de servirle como una experiencia de aprendizaje de sus propios errores.

Pruebas escritas

Aunque la utilización de los exámenes escritos es notablemente superior a la de los orales, no cabe atribuirlo más que a una razón de índole práctica: la facilidad que supone el adaptar el ritmo de evaluación a las posibilidades reales del profesor o incluso, en ciertos tipos de pruebas, la corrección automatizada.

Exámenes de respuesta libre

Este tipo de pruebas se corresponde con los clásicos exámenes donde el alumno redacta su propia respuesta a la pregunta propuesta. Su principal característica es que el alumno debe organizar sus ideas, a la vez que demuestra su capacidad de expresión, seleccionando lo más importante.

En su enunciado debe delimitarse claramente el problema y en ocasiones convendrá precisar la estructura de la respuesta.

La faceta más positiva de este tipo de exámenes es que posibilita apreciar la capacidad para emitir juicios críticos. En su contra tiene dos inconvenientes principales. El primero se refiere al gran tiempo que debe emplearse para su corrección, unido a la dificultad para hacerlo con objetividad, evaluando a todos los alumnos con el mismo criterio, pues intervienen factores externos como el cansancio, el estado de ánimo, la predisposición inicial... Para evitar o al menos reducir este efecto se aconseja:

a) *Leer completamente todos los exámenes para tomar una referencia antes de pasar a la corrección propiamente dicha.*

b) *No realizar la corrección de un examen completo tras otro. Es más adecuado ir corrigiendo pregunta por pregunta todos los exámenes. Así, los mencionados factores externos tienen mayores probabilidades de manifestarse de igual manera para todos los alumnos*[10].

El segundo inconveniente es que limita en demasía el número de parcelas cognitivas a juzgar, no siendo, en muchos casos, la selección de preguntas suficientemente representativa. Esto se debe controlar con una adecuada redacción de las preguntas, intentando que tengan una cierta originalidad e interrelacionando diversos conceptos.

Por todo ello sólo conviene utilizar estas pruebas para evaluar un tipo de actuación que otros métodos no puedan medir, como la síntesis de nociones complejas, comparar dos fenómenos, analizar causas, establecer relaciones...

Exámenes de respuesta abierta y corta

Se trata de una serie de preguntas redactadas de tal modo que exigen por respuesta un concepto predeterminado y preciso.

Es una modificación de las anteriores que evita alguno de sus inconvenientes. Por un lado, permite evaluar un área de conocimientos mayor, ya que se pueden plantear un mayor número de cuestiones que contemplen más aspectos del temario y, normalmente, al ser aceptable una sola respuesta, se gana objetividad en la corrección. Existen estudios que indican que la frecuencia de discordancias debidas únicamente a la contradicción entre dos correctores no alcanza el 2%.

Exámenes tipo test

También conocidas como pruebas PEM (*Preguntas de Elección Múltiple*). Son las denominadas pruebas objetivas[11], que consisten en un cuestionario formado por una serie de preguntas con varias respuestas propuestas, entre las cuales el alumno debe elegir una o varias de ellas.

Entre las principales ventajas de este tipo de pruebas se puede citar las siguientes:

- Evalúan mayor volumen de conocimientos que cualquier otra prueba (siendo factible cubrir la totalidad del temario).
- Permiten graduar la dificultad de la prueba.
- Posibilitan medir procesos intelectuales distintos a la simple memorización.
- Rapidez de corrección, pudiendo realizarse esta de forma automatizada [Mason and Woit, 1998].
- Objetividad.

[10] En todo caso conviene revisar de forma global, en un último paso, aquellos exámenes cuya calificación final sea dudosa por estar en la frontera entre una nota y otra.

[11] Bajo este epígrafe pueden incluirse también otro tipo de pruebas de uso menos frecuente por sus mayores limitaciones, entre las que se encuentran las de proposición incompleta, verdadero-falso, ordenación, localización o asociación.

Sin embargo, tampoco es un método perfecto, presentando los siguientes inconvenientes:

- Su preparación exige mucho tiempo y competencia si se quieren evitar las preguntas arbitrarias, ambiguas o que sólo evalúen el recuerdo.
- Las condiciones teóricas que presentan en sus planteamientos no se corresponden casi nunca con situaciones reales de una forma exacta.
- Resultan costosas si el número de alumnos sobre el que se van a aplicar es reducido.

Situación problemática

Este tipo de pruebas permiten evaluar el conocimiento de los procedimientos, técnicas y herramientas aplicables a un supuesto práctico de características predeterminadas por el docente y semidesarrollado por él mismo.

Las preguntas formuladas sobre el caso deben ser tales que permitan la elaboración de las respuestas en un espacio razonable de tiempo. Presenta ventajas sobre otras pruebas escritas pues, aunque teniendo la finalidad de valorar procesos intelectuales, se acerca más al campo de las habilidades prácticas, pero su calificación es también laboriosa y difícil de objetivar.

Pruebas orales

La prueba oral permite al profesor analizar de forma más detallada los conocimientos reales del alumno y su grado de comprensión, junto con su capacidad de estructuración y ordenación de las respuestas, frente a las cuestiones o problemas planteados.

El hecho de que las pruebas orales hayan decaído en los últimos años puede encontrarse quizás en la masificación de la enseñanza que dificulta el contacto personal y directo. Sin embargo, fuera del ámbito docente, el peso que ha cobrado la técnica de la entrevista en la selección de personal por parte de empresas, apunta hacia una nueva valoración del procedimiento.

Además, como indican algunos estudios, los mismos estudiantes que han dado respuestas aceptables a ciertas preguntas en exámenes escritos, demuestran tener grandes *lagunas conceptuales* al tener que defender los mismos conceptos en una entrevista o en una prueba oral [Greening, 1997], derivándose de esto que muchos alumnos basan su estudio en la elaboración de estrategias para pasar con éxito los exámenes, sin preocuparse de realmente comprender las materias objeto de estudio.

Los mayores inconvenientes que presenta cualquier prueba oral pueden resumirse en los siguientes:

- Estandarización inadecuada. La prueba a que se somete cada estudiante es diferente, lo cual además de romper con la equidad, introduce un factor subjetivo importante en la evaluación.

- Influencia excesiva de factores sobreañadidos, como es la reacción emotiva del alumno y subjetividad del examinador; cada estudiante posee unas características personales que son ajenas a sus conocimientos, como los nervios o su carácter, que pueden perjudicar de forma notable a unos más que a otros, afectando especialmente a su capacidad de reflexión o análisis.
- No permiten cubrir un contenido extenso de la materia.
- Alto coste en tiempo, en relación con el valor limitado de las informaciones que aporta.

Pueden diferenciarse tres tipos de pruebas orales:

- **Exposición autónoma de un tema**. Consiste en el desarrollo de un tema sin que exista interacción entre el expositor y evaluador. Presenta el inconveniente de no poderse profundizar en el conocimiento de determinados conceptos cuya ignorancia se soslaya haciendo una alusión superficial. Sin embargo, permite valorar otra serie de aptitudes como capacidad de estructuración, exhaustividad en el tratamiento o reacción ante un auditorio.
- **Debate**. Su utilización es más frecuente como técnica de enseñanza que como medio de evaluación, por los numerosos sesgos de que puede ser objeto. No obstante, permite valorar no sólo la calidad de la información presentada sino también la oportunidad de la misma y capacidad crítica.
- **Entrevista.** Básicamente existen dos modalidades de entrevistas:
    - *Estructurada*, también denominada interrogatorio, en la cual el alumno ante una pregunta debe proporcionar una respuesta concreta.
    - *Semiestructurada,* cuyo desarrollo es más coherente y distendido. Las preguntas se relacionan con las respuestas anteriores, ganándose fluidez y versatilidad, siendo factible profundizar en temas concretos o pedir justificaciones, así como pasar de los puntos fuertes a los débiles del candidato.

Informes de observación

Este tipo de pruebas implica la observación directa, repetida y estandarizada de la actividad de los alumnos.

Aunque su ventaja teórica sería el permitir una evaluación global, la carencia en la actualidad de medidas objetivas probadas parece aconsejar como razonable el no utilizar este método de simple observación más que con valor de matiz, por el importante componente de subjetividad que conlleva.

3.6.1.2 Evaluación de las habilidades profesionales y de la capacidad de comunicación

Estos dos campos son tradicionalmente olvidados al plantear la evaluación de los estudiantes a pesar de reconocerse que son los que realmente valoran o prueban la capacidad profesional del alumno en función de su competencia para organizar y utilizar datos y técnicas en situaciones reales.

Se desarrollan conjuntamente por disponerse de los mismos instrumentos de medida para su valoración. Instrumentos que, a pesar de su difícil normalización y objetividad, es imprescindible utilizar en cualquiera de sus posibilidades, si se pretende una evaluación completa de los estudiantes.

Prueba real o simulada

Es un tipo de prueba en la cual el estudiante tiene que realizar tareas profesionales, en un medio y condiciones iguales o próximas a aquellas en las que tendrá que desenvolverse en su vida profesional. En los casos de imposibilidad de realización en situación real puede efectuarse en simulación, mediante, por ejemplo, grabación de vídeo u otro sistema directo de recogida de datos.

Se aconseja su utilización cuando la destreza y/o relación interpersonal son el componente principal del objetivo educativo. Sin embargo, presenta dificultades importantes entre las que están: la colaboración de Organizaciones Empresariales o Administración, y la presencia puntual del examinador en el momento en el que el estudiante hace la demostración de las aptitudes requeridas, pues pueden ser períodos prolongados de tiempo.

Realización de un proyecto

Es un tipo de prueba que permite la observación indirecta de una tarea profesional. En ella el estudiante debe realizar una actividad en un período de tiempo variable, pero extenso, que tendrá como resultado un producto que el docente deberá evaluar. El producto puede ser un documento escrito (informe, investigación) o una obra concreta.

Se aconseja utilizar esta técnica si el componente principal del objetivo educativo es una habilidad técnica o intelectual compleja y si se está interesado más en el producto que en el modo de actuar del estudiante.

El objeto del proyecto puede ser elección del alumno, aunque debe ser el profesor quien determine los requisitos mínimos o estructura del producto final. Puede realizarse individualmente o en grupos aunque no se recomienda más de cuatro o cinco personas por grupo. A diferencia de la prueba práctica exige el desarrollo secuencial de todas las etapas que deben realizarse, desde la recogida inicial de información hasta el resultado final.

Las dificultades que presenta su realización son: alto coste en personal docente, por el tiempo que se precisa tanto para su tutela como para evaluar su resultado. A esto hay que añadir la necesidad de instaurar una relación de confianza frente al alumno para evitar el fraude [Carter, 1999].

3.6.1.3 Conclusiones sobre los sistemas de evaluación

A la vista de las ventajas e inconvenientes expuestos para las distintas técnicas de evaluación, y dado que no existe ninguna completa en sí misma, únicamente señalar que, con el fin de conseguir una valoración de la capacidad profesional lo más real posible, siempre ha de plantearse la realización de al menos una prueba de evaluación del campo de los conocimientos y otra de los campos de las habilidades profesionales y capacidad de comunicación.

La determinación del tipo de prueba concreta a realizar en cada campo debe considerar el contexto donde se encuentra para poder ser llevada a cabo.

3.6.2 Calificación

La calificación corresponde a la etapa de la evaluación que tiene por finalidad la interpretación de los resultados obtenidos en la aplicación de los instrumentos de medida diseñados. Cuando varios instrumentos diferentes dan un resultado concordante, a pesar de sus imperfecciones, la fiabilidad de la evaluación aumenta.

Antes de iniciar el proceso de calificación es preciso haber decidido si lo que se pretende de la evaluación es que compare individuos entre sí, o que informe sobre lo que los individuos son capaces o no de realizar.

Pruebas de criterios absolutos y relativos

Son conocidas también como pruebas referidas a criterios y pruebas normalizadas respectivamente.

Una prueba de criterios absolutos es aquella que tiene por finalidad evaluar la actuación de un individuo en relación con un nivel de actuación establecido de antemano basándose en el dominio de un objetivo específico fijado.

Por el contrario las pruebas de criterios relativos tienen por fin el comparar un individuo con otros o con un grupo, sobre la base de la curva de distribución normal de los resultados obtenidos por los estudiantes que han realizado la misma prueba.

Las pruebas de criterios relativos pueden ser válidas para seleccionar un cierto número de individuos, por ejemplo para su admisión a seguir un programa dado. Pero si los objetivos perseguidos son de gran importancia (es decir su dominio o no, tiene consecuencias sobre la colectividad en el ejercicio profesional) es igualmente importante determinar si han sido realmente alcanzados. Esto no puede hacerse

comparando entre sí a los estudiantes, sino comparando la actuación del estudiante con la actuación prevista en el objetivo, es decir, mediante pruebas de criterios absolutos.

El sentido final de las pruebas de criterios absolutos es establecer el mínimo aceptable para certificar si un discente llega al mínimo exigido socialmente para el ejercicio con garantías de su labor profesional. Citando a **R. F. Mager**: "*Poco importan los esfuerzos del estudiante, poco importa que casi haya llegado a la meta... mientras no sea capaz de hacer lo que está obligado a hacer, no se debe certificar que es capaz de hacerlo*" [Mager, 1976].

La evaluación continua debe hacer competir al estudiante consigo mismo y no con otros estudiantes. La lucha debe plantearla contra su ignorancia, pues sería tan injusto el no considerar apto a un estudiante con un nivel de actuación satisfactorio por estar incluido en un grupo de estudiantes particularmente brillante, como peligroso el certificar la aptitud de estudiantes no capacitados, aunque sus resultados sean superiores a la media del grupo al que pertenecen.

Entre las cualidades que debe poseer un sistema de calificación están: *la claridad, la pertinencia, la precisión* y *la objetividad*. Además, para minimizar los factores condicionantes de errores de calificación, conviene tener presentes las siguientes consideraciones o criterios generales:

- Calificación de las respuestas de modo anónimo.
- Si una prueba va a ser corregida por dos o más docentes, antes de la realización de la misma debe haberse acordado el procedimiento de calificación, así como los elementos que debe contener una respuesta completa, realizándose aisladamente la calificación.
- Utilizar un sistema de calificación cuantitativo, según los elementos que se supone deben aparecer en las respuestas. Además, si se considera que para asegurar un nivel de actuación aceptable la respuesta a determinadas preguntas tiene especial relevancia, el peso atribuido a cada una de ellas puede ser diferente y debe especificarse y comunicarse a los alumnos en el momento de la prueba.
- Cuando la corrección no es automatizada, calificar las respuestas de todos los estudiantes a una misma pregunta, antes de pasar a la siguiente.
- No debe formarse un juicio del candidato a partir de una sola pregunta. Para cada estudiante, el cálculo de la calificación será acumulativo y fundamentado en la lectura de varias respuestas de alumnos diferentes, pues contribuye a una mayor seguridad.
- Cuando la evaluación global se efectúa basándose en pruebas diferentes, debe determinarse y comunicarse previamente a los alumnos el peso que cada una de ellas tendrá en la calificación global.

3.7 Fuentes de nuevos conocimientos: La investigación

La función de la Universidad no se limita a la transmisión de conocimientos para la formación de nuevos titulados, sino que debe preparar a los Doctores que constituirán el futuro personal docente e investigador de la Universidad y Centros de Investigación. Estas tareas no se pueden llevar a cabo adecuadamente si no existe una importante labor investigadora en la Universidad.

Quizás el hito por excelencia que, dentro de la comunidad universitaria, se asocia a la labor investigadora es la **Tesis Doctoral**, cuyo significado se encuentra regulado por el Real Decreto 185/1985, donde se exponen los objetivos del tercer ciclo, a saber:

- Disponer de un marco adecuado para la consecución y transmisión de los avances científicos.
- Formar a los nuevos investigadores y preparar equipos de investigación que puedan afrontar con éxito el reto que suponen las nuevas ciencias, técnicas y metodologías.
- Impulsar la formación del nuevo profesorado.
- Perfeccionar el desarrollo profesional, científico, técnico y artístico de los titulados superiores.

De los objetivos presentados se deduce una íntima relación entre docencia e investigación, que se hace más patente en el caso de la Informática en general, y de la Ingeniería del Software en particular. La obsolescencia de esta disciplina es realmente desconcertante. A diferencia de campos más asentados, en esta área se puede ver como sus fundamentos son objeto de importantes cambios, de forma que conceptos que hasta hace apenas seis años eran temas de investigación, se incluyen hoy en día en los temarios de las asignaturas. Esto exige un esfuerzo de actualización constante que requiere de la investigación como fuente de nuevos conocimientos, para facilitar así la docencia cuando ésta coincidiera con el área de investigación desarrollada.

El objetivo de aunar docencia e investigación requiere de una planificación departamental y de una decidida ayuda desde la Universidad. El profesorado es la piedra angular, y son necesarias políticas continuadas de formación de este profesorado que no se acaban con la financiación, sino que requieren de una planificación que determine los objetivos de esta formación. Hay que añadir que la legislación hoy en vigor aumenta la autonomía de los departamentos para confeccionar sus planes de estudio, lo cual lleva indisolublemente pareja una mayor responsabilidad en las oportunidades de formación que se brindan a los alumnos. Esta responsabilidad pasa por planificar asignaturas que permitan abrir las líneas formativas que se desean reforzar. Esto sólo es posible si se cuenta con un profesorado formado, y aquí, la realización de una Tesis Doctoral es una magnifica oportunidad que no se debe desaprovechar [Díaz, 1999].

No obstante, la labor de investigación de un docente no acaba (*o no debiera*) con la culminación de su Tesis Doctoral, sino que, aprovechando la experiencia obtenida en la realización de este trabajo, debe canalizar sus esfuerzos de investigación tanto en la formación de nuevos doctores como en el trasvase de los frutos de su investigación a la sociedad. Es en este último punto donde debe fraguarse la conexión de la comunidad universitaria con el tejido industrial y empresarial donde se encuentre inmersa.

La conexión Universidad - Empresa

La Universidad y la Empresa difieren considerablemente sobre los objetivos e intereses de cada cual en lo tocante a investigación. Sin embargo, la única forma de unificar esfuerzos es acercar posturas entre ambas entidades y abrir caminos para que pueda existir una transferencia entre la investigación realizada en la Universidad y las necesidades actuales de la Empresa [Glass, 1997].

Esta comunicación no sólo se pide en el campo de la transferencia en el contexto de la investigación. De hecho, cada vez son más las voces que abogan porque la formación curricular de los nuevos titulados, especialmente en carreras de carácter técnico como puede ser la Ingeniería Informática, no dé la espalda a las necesidades de la Empresa que ha de acoger a éstos al final de sus estudios.

En este sentido, la Ingeniería del Software es una de las disciplinas sobre las que cae una mayor responsabilidad a la hora de preparar a los estudiantes para su incorporación a la vida laboral [Díaz-Herrera and Powell, 1998], [Villarreal and Butler, 1998], [Wohlin and Regnell, 1999].

A menudo se produce la paradoja de que los estudiantes de Ingeniería Informática han sido educados por profesores que probablemente nunca han trabajado como ingenieros informáticos en ningún puesto. Por otro lado, una parte sustancial de la investigación universitaria que se realiza en cualquier país del mundo desarrollado, la constituye la investigación contratada. Mediante ésta, la empresa encuentra un eficaz apoyo para sus procesos de innovación y desarrollo, y la Universidad aumenta su nivel de calidad en la docencia al aumentar su calidad investigadora y estar en contacto con nuevos procesos o productos útiles. De esta manera, toda la sociedad sale beneficiada.

Por todo ello, la interconexión entre la empresa y la universidad reporta los siguientes beneficios:

- Para la industria, la movilización del potencial intelectual y científico de sus técnicos rompiendo la rutina e incrementando la eficacia de su trabajo. Además, podrá influir en los programas de las titulaciones y tener acceso a las innovaciones en métodos, herramientas y tecnología resultado de la investigación conjunta con el mundo universitario [Beckman et al., 1997].
- Para la Universidad, la valoración de su trabajo y un contacto con la vida real mediante problemas concretos. Así mismo, un beneficio económico que

posibilite la adquisición de nuevos aparatos, productos y becas para estudiantes, que permita a éstos conocer la problemática del desarrollo del software en un entorno de trabajo real [Dawson and Newsham, 1997], [Kuhn, 1998]. También una orientación sobre qué necesita enseñar y cuáles son las necesidades de conocimientos de los futuros ingenieros informáticos.

- Para la sociedad en su conjunto, una mejora en el nivel de efectividad de su Universidad e Industria, que de esta manera pueden ofrecer un mejor servicio a la comunidad.
- En general, un aumento de la reputación y el prestigio de todos los involucrados [Mead et al., 2000].

Una manera de efectuar esta conexión podría ser el intercambio de técnicos altamente cualificados de la Industria y profesores de la Universidad. Los primeros podrían enseñar durante cierto tiempo en la Universidad, y los profesores pasar temporadas en la Industria. Todo esto fomentaría un mayor conocimiento mutuo. De hecho, es práctica común en los países más desarrollados, existiendo algún avance en la Universidad española como puede ser la figura del profesor asociado.

Sin embargo, en un área con un marcado carácter tecnológico, la interacción con el entorno industrial es fundamental por cuanto le ofrece una perspectiva directa de las necesidades que se han de resolver y, por tanto, beneficia de forma inmediata a la labor docente que ha de llevar a cabo un profesor universitario.

Algunas de las Universidades más famosas del mundo, tales como **Harvard** y el **MIT**, tienen acuerdos con compañías para desarrollar programas de investigación académica en áreas específicas donde se esperan descubrimientos de interés práctico. Otro ejemplo es el desarrollo de la biotecnología en Israel. Explotando la investigación en la Universidad, las compañías israelitas están logrando convertir su país en un líder mundial en dicha área. Ejemplos concretos de la colaboración entre la Universidad y la Empresa en asuntos relacionados con la Ingeniería del Software se pueden encontrar en [Beckman, 1999] o en [Beckman et al., 1999].

No obstante, debe mantenerse el respeto mutuo entre la investigación industrial y universitaria. Cada una tiene su finalidad y su papel que desempeñar, siendo ambas complementarias.

Los proyectos de final de carrera

Según se contempla en los planes de estudio de Ingeniería Técnica de Informática e Ingeniería Informática de la Universidad de Salamanca y para la obtención del correspondiente título, los alumnos deben realizar un Proyecto Fin de Carrera. En concreto, para los alumnos del Ingeniería Técnica se trata de una asignatura obligatoria que tiene asignados 9 créditos prácticos y para los de Ingeniería Informática es troncal de 6 créditos prácticos. La realización de estos proyectos puede considerarse a veces como investigación aplicada. Es por ello por lo que se ha considerado la inclusión en este proyecto docente de un apartado sobre este tema.

En particular, el Proyecto Fin de Carrera consistirá en un diseño, aplicación, explotación y gestión de sistemas informáticos o en un estudio sobre algún tema o tecnología avanzada que, por su novedad o escasa implantación, no haya sido objeto de un estudio detenido en las asignaturas correspondientes de la carrera.

En general, en estos trabajos es donde mejor se conjugan los objetivos de la Universidad: *docencia* e *investigación*.

Para el alumno, el desarrollo de un proyecto supone, por una parte, la consecución de una formación más específica en un determinado tema, y por otra, la utilización de métodos y medios propios de un entorno profesional. En este sentido, el trabajo constituirá la prueba final de madurez antes de pasar al desarrollo de su etapa profesional.

Al docente, le permitirá renovar y ampliar conocimientos, de forma continua, sobre los temas impartidos, así como disponer de un punto de vista práctico de los conceptos teóricos que trata en clase. Este último punto resulta especialmente significativo e importante en una Ingeniería.

Para conseguir el mayor fruto de estos trabajos, es imprescindible que el alumno encuentre un ambiente adecuado tanto en el aspecto científico como en el humano, así como los medios materiales necesarios para su desarrollo. En este sentido, su financiación puede provenir, a parte de la propia Universidad, de la colaboración con la Industria. En la actualidad existe una activa relación Universidad-Empresa materializada en convenios de cooperación educativa que permiten acercar al alumno al mundo laboral y la realización de su Proyecto Fin de Carrera.

Por otra parte, es interesante que los proyectos no se reduzcan a la obtención de resultados teóricos, sino que todos ellos contemplen una parte de experimentación y diseño práctico en función de dichos resultados. Esta sería la forma de vincular la investigación básica con la aplicada, prestando especial atención a esta última ya que los proyectos se desarrollan dentro de una Ingeniería. Es muy interesante para el universitario tener una experiencia investigadora, que complete su formación. Los

Proyectos Fin de Carrera, además, pueden constituir un elemento más, de cierta importancia, dentro de la actividad investigadora de los Departamentos.

Los proyectos suponen, por tanto, una oportunidad de enriquecimiento mutuo entre Universidad y alumno, que rompa con la tradicional y negativa idea que se puede tener de una docencia universitaria que solamente esté basada en las clases teórico-prácticas.

3.8 Aseguramiento de la calidad de la docencia

La calidad es un objetivo fundamental para los directivos de una empresa junto a los dos parámetros clásicos de su gestión: el dinero y el tiempo. El mercado actual es sumamente competitivo; no basta con producir masivamente los productos o servicios, vender es lo importante y sólo se produce un producto cuando se tiene la seguridad de aceptación por parte del cliente. Por tanto, lo realmente importante es satisfacer al cliente, conocer sus necesidades para luego definirlas en forma de requisitos a cumplir.

En la vida cotidiana, la calidad representa las propiedades inherentes a un objeto, de forma que pueda ser comparado con otros objetos de su especie para determinar si es mejor, igual o peor. Calidad es sinónimo de bondad, excelencia o superioridad.

Para la comunidad universitaria, la calidad debe ser un objetivo tan importante como lo es para la empresa. El profesor de Universidad debe ver en la calidad un camino hacia la excelencia, tanto en su actividad investigadora como en su actividad docente. El antiguo papel del profesor como mero conferenciante y proveedor de conocimiento debe ser reemplazado por un nuevo rol: *el profesor como comunicador, consejero y gestor de la clase* [Null, 1996].

Centrando la atención en la actividad docente, la calidad puede verse como una actividad contractual donde la Universidad es la "*empresa*" y los clientes son varios. El cliente más directo es el estudiante, el producto que se le ofrece debe ser un currículo que cumpla unos requisitos acordes a los objetivos marcados por la titulación elegida. La sociedad es el cliente indirecto de la Universidad, donde el producto que se le ofrece son titulados formados y preparados para introducirse en el mundo real y rendir de acuerdo con las necesidades de esa sociedad. Por último, el propio docente será a la vez mecanismo y cliente de la Universidad, al buscar por un lado la excelencia de conocimiento en una determinada materia y por otro lado la satisfacción personal conseguida cuando los estudiantes salen cumpliendo los requisitos de conocimientos que la Universidad marca y la adecuación que la sociedad demanda.

Para seguir las directrices de la mejora continua hacen falta dos elementos imprescindibles: en primer lugar se requiere de *una apuesta firme por unos servicios de calidad por parte de los órganos institucionales (Departamento, Facultad, Rectorado) que rigen directa o indirectamente la actividad docente del profesor,* y por otra parte de

una actitud personal favorable del propio docente en relación con la calidad en la docencia.

La primera premisa se satisface en la Universidad de Salamanca por la apuesta del equipo rectoral actualmente vigente, y que se refleja en su programa electoral [Berdugo, 1998] bajo un conjunto de medidas destinadas a potenciar los *"programas de calidad de la docencia*", a saber:

- Exigencia en el cumplimiento del calendario docente.
- Introducción de sistemas de tutoría activa.
- Consolidación del programa de adquisición y ampliación de infraestructura para prácticas de laboratorio, así como la ampliación del programa de prácticas de campo.
- Potenciar las prácticas realizadas fuera de la Universidad, concertadas con empresas, instituciones y colegios profesionales.
- Seguir una política de aumento y mejora de la gestión en las asignaturas de libre disposición.
- Introducción de sistemas de evaluación de los alumnos que tengan en cuenta el rendimiento global y garanticen el uso de criterios y objetivos iguales para todos los alumnos en cada materia.
- Organización de sistemas de coordinación, seguimiento y mejora de los programas docentes en cada curso y en cada carrera.
- Programas de estímulo al uso de tecnologías avanzadas en la actividad docente.
- Apoyo a las iniciativas de formación del profesorado.
- Potenciar la participación de toda la comunidad universitaria en programas de evaluación y mejora de la calidad de la enseñanza.

La actitud personal hacia una política de calidad en docencia puede verse reflejada de muy diversas formas, con diferentes actividades realizadas de forma individual o colectiva. Es importante que esta actitud hacia la calidad sea algo voluntario, no impuesto, para que realmente se obtengan los beneficios buscados.

A continuación se van a comentar las actividades más destacadas que, desde la experiencia personal, he llevado a cabo para incidir en la mejora continua de mi labor docente y que se pueden resumir en tres apartados: *el plan de calidad, las tutorías activas* y *las experiencias de evaluación por pares.*

3.8.1 El plan de calidad

La motivación principal para elaborar un plan de calidad de una asignatura es la percepción personal, por parte del profesor responsable (o profesores) de la misma, de que se desea comenzar un proceso de mejora continua en la docencia de una o varias materias. Este proceso de mejora continua debe comenzar con un plan de actuación que establezca como primer objetivo el conocimiento profundo del desarrollo docente de la asignatura o asignaturas elegidas (*como experiencia inicial se recomienda comenzar el proceso de mejora continua con una única asignatura*).

Una forma adecuada de establecer un mayor grado de control sobre la asignatura es ampliar el documento de planificación docente de la asignatura a un plan de calidad, en el que principalmente se introduzcan dos nuevos aspectos [García et al., 1999a]:

- ***Una lista de objetivos***. Lista que debe verse completada con una serie de líneas de acción para lograr dichos objetivos. Cada una de estas líneas de acción debe quedar completamente definida mediante un marco formado por cuatro entradas:
    - *Cuándo:* Atributo temporal que indique el momento aproximado en el que ha de llevarse a cabo la línea de acción.
    - *Quién:* Faceta que indica los involucrados en la realización y éxito de la línea de acción.
    - *Medios:* Característica que representa los elementos, normalmente materiales, con los que se ha de contar para la culminación de la línea de acción.
    - *Evaluación:* Es imprescindible establecer un medio para evaluar cada una de las líneas de acción establecidas.

  Los objetivos pueden clasificarse en dos categorías ortogonales: por un lado *los objetivos que se pretenden lograr con la asignatura en sí*; y por otro lado *los objetivos personales del propio profesor como docente*.
- ***Una realimentación del proceso de evaluación***. Del análisis de los resultados de la evaluación de cada una de las líneas de acción, se debe obtener una valoración crítica de la asignatura que permita un nuevo planteamiento para el siguiente curso. Esta valoración crítica se puede centrar en la consecución de los tres apartados siguientes:
    - *Puntos fuertes*: Se deben detectar las parcelas de la asignatura donde se haya conseguido un mayor impacto sobre los objetivos, con el fin de potenciarlos y mantener su nivel de influencia.
    - *Áreas de mejora*: La detección de aquellas partes de la asignatura que más se han alejado de los objetivos, o aquellos objetivos que no

han dado el fruto esperado, es un punto esencial en el proceso de mejora continua porque marca las áreas de actuación de cara al nuevo curso, donde la experiencia obtenida y contrastada por los medios de evaluación establecidos es fundamental y, por sí sola, justifica la realización del plan de calidad.

- *Redefinición del plan de calidad para el curso siguiente*: Actividad que coincide con la realización del plan de calidad para el curso siguiente.

Según lo visto se puede afirmar que el proceso de creación de un plan de calidad para una asignatura es un proceso iterativo que se alimenta de la experiencia obtenida en las iteraciones previas, tal y como se muestra en la Figura 3.3.

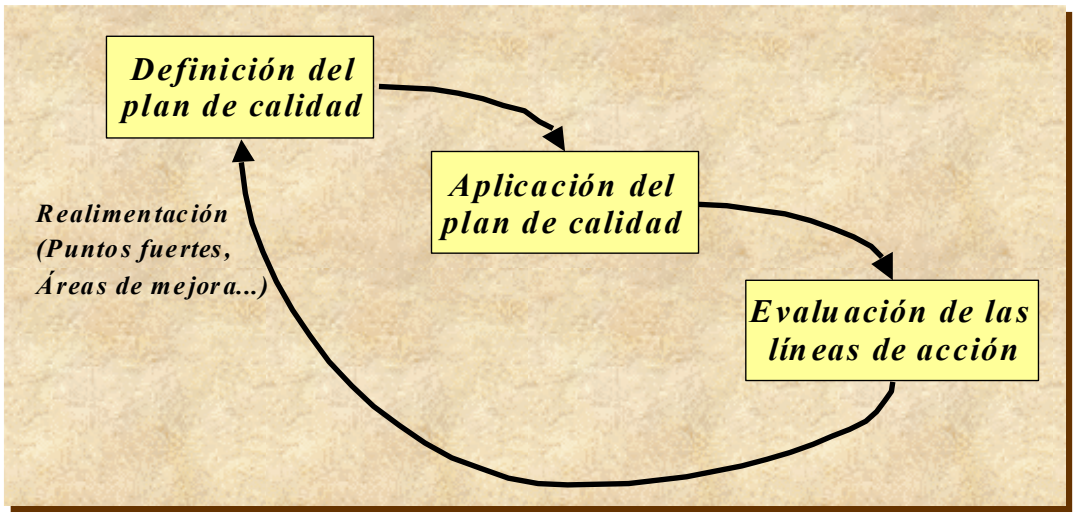

Figura 3.3. Proceso de creación del plan de calidad

El esqueleto básico del plan de calidad queda reflejado en la Figura 3.4.

Portada

Lista de cambios

Tabla de contenidos

1. Introducción
2. Objetivos de la asignatura
3. Objetivos personales
4. Otras actividades
5. Resultado de la evaluación
6. Programa de la asignatura

Anexo. Resumen de los objetivos

Figura 3.4. Esquema de la composición del plan de calidad

Experiencias prácticas en la aplicación del plan de calidad

La realización y puesta en práctica de un plan de calidad para la asignatura de Análisis e Ingeniería del Software se abordó por primera vez en el curso 1996-1997 en la Ingeniería Técnica en Informática de Gestión de la Universidad de Burgos [García, 1996].

Esta experiencia piloto nació como una inquietud personal del responsable de la asignatura por establecer un plan de acción que le permitiese obtener unas conclusiones sobre los métodos docentes que se aplicaban en una asignatura tradicionalmente árida, como es la Ingeniería del Software.

Los objetivos iniciales de esta iniciativa fueron principalmente dos: *definir la estructura del plan de calidad* y *establecer los medios de evaluación de las líneas de acción*; en este sentido destaca la confección de un cuestionario de evaluación destinado a los alumnos de la asignatura.

Estas labores iniciales fueron realizadas en colaboración con la Unidad Técnica de Calidad de la Universidad de Burgos, desde donde se guiaron todos los pasos dados.

Los resultados de esta primera experiencia fueron muy positivos, especialmente por el mayor conocimiento y domino que se adquiere de la asignatura impartida, y no se está haciendo referencia exclusivamente a los aspectos cognitivos de la misma. Los datos obtenidos permitieron establecer los puntos fuertes y, lo que era más importante, las áreas de mejora de la asignatura. Cabe destacar como una de las principales áreas de mejora detectadas la necesidad de definir más medios de control/evaluación aparte del cuestionario realizado.

Esta experiencia inicial sirvió para la creación del nuevo plan de calidad para la asignatura de Ingeniería del Software en el curso 1997-1998 [García, 1997a], pero se amplió la iniciativa a otra asignatura, Programación Avanzada [García, 1997b], que era la primera vez que se impartía en el plan de estudios de esta Ingeniería Informática.

La creación y puesta en marcha de una plan de calidad en una asignatura es siempre interesante y positivo, pero sus beneficios potenciales aumentan en los casos de asignaturas de nueva implantación, porque esta técnica ayuda a detectar los numerosos conflictos y problemas que tienen las asignaturas en sus inicios.

En el curso 1998-1999, se aplicaron las experiencias anteriores en la realización del plan de calidad de la asignatura Ingeniería del Software, pero dentro de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca [García, 1999]. De nuevo, la creación de un plan de calidad para esta asignatura sirvió de gran ayuda para el profesor, por encontrarse esta materia en un momento de transición hacia un nuevo plan de estudios que ha comenzado a impartirse el curso 1999-2000; además dicha transición también estaba condicionada por el comienzo del Segundo Ciclo de la

Ingeniería Informática, donde la materia de Ingeniería del Software también está presente.

En el presente curso se va acometer la tarea de la realización de un plan de calidad para el bienio 1999-2001, pero no centrado en una única asignatura, sino que abarque una unidad docente completa, la **Unidad Docente de Ingeniería del Software y Orientación a Objetos** [García et al., 2000], que tiene competencias en dos titulaciones universitarias diferentes, aunque relacionadas: la *Ingeniería Técnica en Informática de Gestión (Plan de 1997) y la Ingeniería Informática*, ambas impartidas en la Universidad de Salamanca.

3.8.2 Las tutorías activas

Por acuerdo de la Junta Ordinaria de la Facultad de Ciencias de 9 de diciembre de 1998 se recomienda entre las medidas prioritarias, la implantación de un sistema de tutorías activas para todas las titulaciones de dicha Facultad. Esta iniciativa entra en vigor en el curso 1999-2000 [USAL, 1999].

Podrán ser tutores activos los profesores ordinarios, los ayudantes doctores y los asociados a tiempo completo, también doctores, que de forma voluntaria deseen participar en el proyecto y tengan un buen conocimiento del plan de estudios de la carrera. Los tutores serán nombrados por la Junta de Facultad e incluidos en la programación docente anual.

La función del tutor será orientar e informar al estudiante en cualquiera de los aspectos académicos relacionados con su estancia en la Universidad, en especial en lo relativo a la organización de su recorrido curricular. En concreto, su tarea de ayuda y consejo se referirá, al menos, a la elección de asignaturas durante el proceso de matricula y a la planificación de los estudios, particularmente en lo tocante a la orientación profesional elegida por el alumno.

Todos los alumnos tienen derecho a la asignación de un tutor, aunque no están obligados a recibir su asesoramiento. Cada tutor podrá tutelar a un máximo de quince alumnos. La asignación de alumnos a los tutores se hace en el primer curso de forma aleatoria; asignación que se mantendrá, en principio, durante todo el primer ciclo.

El tutor convocará a cada uno de los alumnos que tenga asignados al menos dos veces al año, una cada semestre. Asimismo, si el tutor lo considera conveniente o cualquiera de los alumnos lo solicitara, podrán celebrarse otras entrevistas en cualquier momento del curso.

En esta primera experiencia con las tutorías activas, acaecida en el presente curso académico, me han sido asignados doce alumnos de primer curso de la Ingeniería Técnica en Informática de Sistemas.

3.8.3 Experiencias de evaluación por pares

La evaluación, que es un punto imprescindible en un proceso de mejora continua porque permite conocer el grado en que una actividad o un programa de calidad se ajusta a los objetivos preestablecidos [Balbin, 1999], se suele convertir en la causa más común de reticencia a la implantación de la calidad dentro de un colectivo, tendiendo a confundir el proceso de evaluación como "*una caza de brujas*" o encontrando en él un carácter sancionador, cuando en realidad, como indica el profesor **J. M. Manso** [Manso, 1997], lo que se está buscando es la evaluación de las actividades, no de las personas.

Los métodos de evaluación de un proceso de calidad en general, y particularmente en el apartado de la docencia universitaria, se pueden dividir en *métodos de autoevaluación o evaluación interna* y en *métodos de evaluación externa* [CU, 1997].

Los métodos de autoevaluación o evaluación interna son aquellos que son responsabilidad del individuo o del grupo en que éste está inmerso. Este tipo de métodos de evaluación es muy importante para el seguimiento continuado del proceso de mejora continua que se esté realizando, dado que la cercanía de los recursos necesarios para llevar a cabo la evaluación no requiere de una gran inversión económica ni de tiempo. No obstante, el proceso de mejora continua en el apartado de la calidad se presta a ser refrendado por una evaluación externa al entorno en donde tiene lugar. Los métodos de evaluación externa tienen el inconveniente de que la disponibilidad de los recursos necesarios (*principalmente recursos humanos en su papel de evaluadores*) para su puesta en marcha requieren de un desembolso económico más grande, así como una mayor planificación entre evaluador(es) y evaluado(s).

A parte del sistema de evaluación del profesorado que sigue la Universidad de Salamanca en su *Plan para la Evaluación y Mejora de la Calidad en la Enseñanza*, personalmente se ha participado en otras experiencias relacionadas con la evaluación, tanto interna como externa.

En este sentido, aunque existen varios los métodos para la evaluación de la calidad de la docencia [Carbone and Kaasbρll, 1998], se optado por la utilización del protocolo de evaluación externa, denominado evaluación por pares, que se describe en [García et al., 1998], [García et al., 1999b], para la evaluación externa de la actividad docente en asignaturas del perfil de la plaza a concurso, con unos resultados altamente positivos.

A continuación, y a forma de esquema, se enumeran las fases de que consta el protocolo de evaluación por pares [García et al., 1999b]:

1. *Inicio y planificación del proceso de evaluación por pares.*
2. *Redacción de una memoria sobre la asignatura por parte de su responsable.*
3. *Estudio de la memoria por parte del evaluador.*
4. *Entrevista del evaluador con alumnos que hayan cursado la asignatura.*

5. *Informe provisional del evaluador.*
6. *Entrevista entre evaluador y evaluado.*
7. *Informe final del proceso de evaluación.*
8. *Reunión para la **autoevaluación** del proceso de evaluación.*
9. *Aplicación de los resultados del proceso de evaluación en la nueva revisión del plan de calidad para la asignatura evaluada.*

3.9 Referencias

[Balbin, 1999] Balbin, Isaac. "*Is Your Degree Quality Endorsed?*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 60-63. ACM. 1999.

[Beard, 1974] Beard, R. "*Pedagogía y Didáctica en la Enseñanza Universitaria*". Oikos- Tau, 1974.

[Beckman, 1999] Beckman, Kathy. "*Directory of Industry and University Collaborations with a Focus on Software Engineering Education and Training, Version 7*". Special Report CMU/SEI-99-SR-001, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). February, 1999.

[Beckman et al., 1997] Beckman, Kathy, Khajenoori, Soheil, Coulter, Neal and Mead, Nancy R. "*Collaborations: Closing the Industry-Academia Gap*". IEEE Software, 14(6):49-57. November/December, 1997.

[Beckman et al., 1999] Beckman, Kathy, Lawrence, Jimmy, Mead, Nancy, O'Mary, George, Parish, Cynthia and Walker, Hope. "*Industry/University Collaborations: Different Perspectives Heighten Mutual Opportunities*". Software Engineering Institute. http://www.sei.cmu.edu/topics/collaborating/ed/indust-univ-collabs.html. [Última vez visitado, 18/10/1999]. August, 1999.

[Berdugo, 1998] Berdugo Gómez de la Torre, Ignacio. "*Candidatura a Rector Encabezada por Ignacio Berdugo Gómez de la Torre. Programa Electoral*". Universidad de Salamanca, 1998.

[Bloom, 1956] Bloom, B. "*Taxonomy of Educational Objectives: Handbook I: Cognitive Domain*". David McKay, 1956.

[Carbone and Kaasbøll, 1998] Carbone, Angela and Kaasbøll, Jens J. "*A Survey of Methods Used to Evaluate Computer Science Teaching*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 41-45. ACM. 1998.

[Carter, 1999] Carter, Janet. "*Collaboration or Plagiarism: GAT Happens when Students Work Together*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 52-55. ACM. 1999.

[Crawford and Fekete, 1997] Crawford, Kathryn and Fekete, Alan. "*What Exams Results Really Measure?*". In Proceedings of the second Australasian conference on Computer science education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. Pages 185-190. 1997.

[CU, 1997] Consejo de Universidades. "*Plan Nacional de la Calidad de las Universidades. Guía de Evaluación*". Secretaría General del Consejo de Universidades, 1997.

[Davis et al., 1997] Davis, Gordon B., Gorgone, John T., Couger, J. Daniel, Feinstein, David L. and Longenecker, Jr. Herbert E. (Editors) "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS and AITP, 1997.

[Dawson and Newsham, 1997] Dawson, Ray and Newsham, Ron. "*Introducing Software Engineers to the Real World*". IEEE Software, 14(6):37-43. November/December, 1997.

[Díaz, 1999] Díaz, Oscar. "*¿Para Qué la Tesis Doctoral?*". Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISDB'99). (Cáceres, 24-25 de noviembre de 1999). Páginas 3-6. 1999.

[Díaz-Herrera, 1998] Díaz-Herrera, Jorge and Powell, Gerald M. "*Educating Industrial-Strength Software Engineers*". In Proceedings of the 11th Conference on Software Engineering Education and Training (CSEE&T '98). (February 22-25, 1998. Atlanta, GA – USA). IEEE Computer Society. Pages 139-150. 1998.

[García, 1996] García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1996-1997. 1996.

[García, 1997a] García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

[García, 1997b] García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Programación Avanzada*". Tercer Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

[García, 1999] García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Ingeniería del Software*". Tercer Curso de la Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Curso 1998-1999. 1999.

[García et al., 1998] García Peñalvo, Francisco José, Montero García, Eduardo y Arranz Val, Pablo. "*Proceso de Evaluación por Pares. Una Experiencia Práctica*". En las actas de las II Jornadas de Calidad y Universidad: Calidad en la Docencia (Burgos, 10-11 de Noviembre de 1998).

[García et al., 1999a] García Peñalvo, Francisco José, Moreno García, María N., González Talaván, Guillermo y Moreno Montero, Ángeles María. "*Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 46 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 1999b] García Peñalvo, Francisco José, Moreno García, María N., Montero García, Eduardo y Arranz Val, Pablo. "*Evaluación del Profesorado: Un Protocolo de Evaluación por Pares*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 47 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 2000] García Peñalvo, Francisco José, Moreno García, María N., García-Bermejo Giner, José Rafael y Luis Reboredo, Ana de. "*Unidad Docente de Ingeniería*

del Software y Orientación a Objetos. Plan de Calidad Versión 1.1". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García Carrasco, et al., 1999] García Carrasco, Joaquín, García del Dujo, Ángel, López Fernández, Ricardo, Mompó Gómez, Rafael, Navazo Suela, María Agustina, Pérez Juárez, María Ángeles, Redoli Granados, Judith, Regueras Santos, Luisa María, Rodríguez Pajares, Blanca y Verdú Pérez, María Jesús. "*Nuevas Tecnologías y Formación*". PCWEEK. Editorial América Ibérica, Madrid. 1999.

[Glass, 1997] Glass, Robert L. "*Revisiting the Industry/Academe Communication Chasm*". Communications of the ACM, 40(6): 11-13. June, 1997.

[Gómez, 1981] Gómez Pérez, R. Prólogo de la obra de J. Pujol y J. P. Fons. "*Los Métodos de Enseñanza Universitaria*". Eunsa. 1981.

[González, 1996] González Casal, J. "*Estudio de la Profesión Informática en España durante 1995*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (29):13-16. 1996.

[Greening, 1997] Greening, Tony. "*Examining Student Learning of Computer Science*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 63-66. ACM. 1997.

[Gullbert, 1989] Gullbert, J. J. "*Guía Pedagógica*". Organización Mundial de la Salud. Quinta Edición. Editado por Instituto de Ciencias de la Educación. Universidad de Valladolid. 1989.

[Hale, 1964] Hale. "*Reports of the Committee on University Teaching Methods"*. Londres, University Grants Committee, 1964.

[Kuhn, 1998] Kuhn, Sarah. "*The Software Design Studio: An Exploration*". IEEE Software, 15(2):65-71. March/April, 1998.

[Lafourcade, 1974] Lafourcade, P. "*Planteamiento, Conducción y Evaluación de la Enseñanza Universitaria*". Kapeluzs, 1974.

[Lara, 1997] Lara Ortega, Fernando. "*Principios de Calidad en la Docencia*". Actas de las I Jornadas sobre Calidad y Universidad. Universidad de Burgos. Noviembre, 1997.

[Lloyd, 1968] Lloyd, D. H. "*A Concept of Improvement of Learning Response in the Taught Lesson*". Visual Education, 1968.

[Mager, 1976] Mager, R. F. "*Creación de Actitudes y Aprendizajes*". 2ª edición. Colección Biblioteca del Educador. Editorial Marova. 1976.

[Manso, 1997] Manso Martínez, J. M. "*Docencia en la Universidad: Lo que Es y lo que Debe Ser*". Actas de las I Jornadas sobre Calidad y Universidad – Hacia una Universidad de Calidad. Burgos, 10-13 de Nov. de 1997.

[Marchionini and Maurer, 1995] Marchionini, Gary and Maurer, Hermann. "*The Roles of Digital Libraries in Teaching and Learning*". Communications of the ACM, 38(4):67-75. April, 1995.

[Mason and Woit, 1998] Mason, David V. and Woit, Denise M. "*Integrating Technology into Computer Science Examinations*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 140-144. 1998.

[Mead et al., 2000] Mead, Nancy, Unpingco, Perla, Beckman, Kathy, Walker, Hope, Parish, Cynthia and O'Mary, George. "*Industry/University Collaborations. Different Perspectives Heighten Mutual Opportunities*". Crosstalk, The Journal of Defense Software Engineering, 13(3):10-15. March, 2000.

[Mercuri, 1998] Mercuri, Rebecca. "*In Search of Academy Integrity*". Communications of the ACM, 40(5):136. May, 1998.

[Mora et al., 1995] Mora Núñez, N., Prima Rodríguez, M., González López, R., Crespo Faja, F. y Díaz Vázquez, J. E. "*Propuesta de Organización Docente para Asignaturas Troncales de Pocos Créditos, Basada en Principios Constructivistas*". Actas de las III Jornadas Universitarias sobre Innovación Educativa en las Enseñanzas Técnicas (Ferrol – A Coruña. Septiembre, 1995). Tomo II. Páginas 285-292. 1995.

[Neumann, 1998] Neumann, Peter G. "*Risks of E-Education*". Communications of the ACM, 40(10):136. October, 1998.

[Nishida et al., 1996] Nishida, Tomohiro, Saitoh, Akinori, Tsujino, Yoshihiro and Tokura, Nobuki. "*Lecture Supporting System by E-mail and WWW*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 280-284. ACM. 1996.

[Null, 1996] Null, Linda. "*Applying TQM in the Computer Science Classroom*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 120-124. ACM. 1996.

[Orden, 1985] Orden Hoz, A. de la. "*Modelos de Evaluación Universitaria*". Revista Española de Pedagogía. Nº 169. 1985.

[Paxton, 1996] Paxton, John T. "*Webucation: Using the Web as Classroom Tool*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 285-289. ACM. 1996.

[Pozo, 1982] Pozo Pardo, A. del. "*La Didáctica de Hoy*". Hijos de Santiago Rodríguez, Burgos, 1982.

[Riser and Gotterbarn, 1998] Riser, Robert and Gotterbarn, Donald. "*On-line Journal: A tool for enhancing Student Journals*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 203-205. ACM. 1998.

[Rodríguez, 1980] Rodríguez Dieguez, J. "*Didáctica General*". Cincel, 1980.

[Sanchís y Torralba, 1997] Sanchís Marco, F. y Torralba Martínez, J. M. "*Seguimiento del Mercado Laboral como Guía para los Diseños Curriculares. El Caso de las Ingenierías Informáticas*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (31):18-23. 1997.

[Ullman and Widom, 1997] Ullman, Jef and Widom, Jennifer. "*A First Course in Database Systems*". Prentice-Hall, 1997.

[UNESCO, 1973] UNESCO. "*Aprender a Ser*". Informe de la Comisión Internacional para el Desarrollo de la Educación. Editorial Alianza, Madrid. 1973.

[USAL, 1999] Universidad de Salamanca. "*Guía Académica Curso 1999-2000. Facultad de Ciencias*". Facultad de Ciencias - Universidad de Salamanca. 1999.

[Veraart and Wright, 1996] Veraart, V. E. and Wright, S. L. "*Supporting Software Engineering Education with Local Web Site*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 275-279. ACM. 1996.

[Vetter and Severance, 1997] Vetter, Ronald J. and Severance, Charles. "*Web-Based Education Experiences*". IEEE Computer, 30(11):139-141. November, 1997.

[Villarreal and Butler, 1998] Villarreal, E. E. and Butler, D. "*Giving Computer Science Students a Real-World Experience*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 40-44. 1998.

[Wohlin and Regnell, 1999] Wohlin, Claes and Regnell, Björn. "*Achieving Industrial Relevance in Software Engineering Education*". In Proceedings of the 12th Conference on Software Engineering Education and Training (CSEE&T '99). (22 - 24 March, 1999. New Orleans, Louisiana – USA). IEEE Computer Society. Pages 16-25. 1999.

Capítulo 4
Definición del Proyecto Docente

Es un hecho ampliamente asumido que la Informática es hoy en día un factor social de gran relevancia. El objetivo de la titulación donde se circunscribe la plaza objeto de concurso, la *Ingeniería Técnica en Informática de Sistemas*, es la formación de profesionales (*ingenieros técnicos en Informática*) que se puedan incorporar a un mercado laboral, actualmente en plena demanda de estos titulados, con unas garantías plenas de calidad en cuanto a la función que deberán desempeñar en sus puestos de trabajo.

Actualmente, y a consecuencia de la falta de madurez que todavía sufre la Informática, es difícil precisar qué se entiende por un ingeniero informático, ya sea técnico o superior. Esta bruma que rodea a la figura de los informáticos es aprovechada por multitud de personas, ya sean tituladas o no, para acogerse a puestos laborales donde se interacciona de una manera u otra con un computador. Es por este motivo por el que se debe hacer un esfuerzo por diferenciar a los *ingenieros en Informática*[12] de los informáticos en general.

En este sentido, y con un ámbito internacional, en 1998 se formó el ***Software Engineering Cordinating Committee*** (SWECC) bajo el auspició de **IEEE-CS** y **ACM**. Su misión es cuidar la evolución de la Ingeniería del Software como una disciplina profesional dentro del mundo de la Informática. Para lo cual se pretende documentar el cuerpo de conocimientos de la disciplina, recomendar un criterio de acreditación de los titulados, desarrollar un modelo de currículo, mantener un código ético y definir un conjunto de estándares. Actualmente, los proyectos que se encuentran en marcha son:

[12] Como ya se ha comentado antes en este proyecto docente, cada vez existe una mayor tendencia en el ámbito internacional a denominar Ingeniería del Software a la titulación que en España equivaldría a la Ingeniería Informática, sin entrar en las diferenciaciones entre titulados de primer o segundo ciclo en nuestro país.

- ***Software Engineering Body of Knowledge (SWEBOK) – Cuerpo de conocimiento de la Ingeniería del Software:*** Tiene como objetivos *clarificar y definir los límites de la ingeniería del software con respecto a otras disciplinas* y *ofrecer los fundamentos para el desarrollo de una propuesta curricular y el material para la certificación de los profesionales.* Este proyecto ha obtenido como resultado la guía con el cuerpo de conocimiento de Ingeniería del Software, que tras pasar por la denominada **Straw Man Version** (*versión de hombre de paja*) [Bourque et al., 1998], actualmente se encuentra en la **Stone Man Version v0.6** (*versión del hombre de piedra*) [Abran et al., 2000] y se pretende que en el año 2001 se llegue a la **Iron Man Versión** (*versión del hombre de hierro*).
- ***Software Engineering Education Project – Proyecto de Educación en Ingeniería del Software [ACM/IEEE-CS, 1999a]:*** Reúne dos subproyectos, *un modelo de acreditación para programas universitarios* [ACM/IEEE-CS, 1998] y *el plan para el proyecto de educación en Ingeniería del Software (Plan for the Software Engineering Education Project – SWEEP)* [ACM/IEEE-CS, 1999b]. Hasta mediados del año 2000 no se tendrá una propuesta curricular sobre en el campo de la Ingeniería del Software.
- ***Software Engineering Code of Ethics and Professional Practice – Código de Ética y Práctica Procesional de Ingeniería de Software [ACM/IEEE-CS, 1999c]:*** El código ético de la profesión fue aprobado por las asociaciones **ACM** e **IEEE-CS** en su versión 5.2 [ACM/IEEE-CS, 1999c][13] como el estándar para la enseñanza y la práctica de la Ingeniería del Software. Este código ha sido desarrollado por un grupo dirigido por **Donald Gotterbarn,** siendo propuesto tras varias versiones[14] y después de revisar los códigos éticos de otras sociedades. El código ético contiene ocho principios relacionados con el comportamiento y las decisiones tomadas por los profesionales. Un breve resumen del mismo se presenta en el Cuadro 4.1.

Tras la presentación de los esfuerzos en pro de la definición de la profesión de *ingeniero informático* o *ingeniero del software* realizados por ACM e IEEE-CS, se vuelve a centrar la atención en el contexto nacional, y más concretamente en el de la Universidad de Salamanca, para definir de una forma más precisa el perfil de los titulados en *Ingeniería Técnica en Informática de Sistemas*; de forma que éstos sean profesionales que *utilicen un sólido conjunto de fundamentos de Ciencias de la Informática para solventar problemas reales, interaccionando con clientes y usuarios,*

[13] En [Dolado, 1999] el **Dr. D. Javier Dolado**, de la Universidad de San Sebastián, ha traducido el código ético en su versión 5.2 al español.

[14] La versión del código ético más conocida, con anterioridad a la aprobación de la versión 5.2, fue la versión 3.0 [Gotterbarn et al., 1997a], [Gotterbarn et al., 1997b]. Precisamente en [Gotterbarn et al., 1999a] y en [Gotterbarn et al., 1999b] en se comenta la versión 5.2 comparándola con la versión 3.0.

debiendo hacer uso una capacidad de comunicación oral y escrita correcta y fluida, y actuando siempre de acuerdo al código ético marcado por su profesión.

Los códigos éticos tienen una función esencial para caracterizar una profesión, y para que una disciplina adquiera el carácter de profesión debe poseer un código de conducta.

Se pueden resumir las principales funciones de los códigos éticos en los siguientes apartados [Bowyer, 1996]:

1) Simbolizar una profesión.
2) Proteger los intereses del grupo.
3) Inspirar buena conducta.
4) Educar a los miembros de la profesión.
5) Disciplinar a sus afiliados.
6) Fomentar las relaciones externas.
7) Enumerar los principios morales básicos.
8) Expresar los ideales a los que se debe aspirar.
9) Mostrar las reglas básicas de comportamiento
10) Ofrecer guías de comportamiento.
11) Enumerar derechos y responsabilidades.

Los códigos de conducta van más allá de la pura normativa legal, ya que ayudan a guiar el comportamiento en multitud de situaciones para las que no existe referencia legal.

El código propuesto por ACM y IEEE-CS tiene como objetivo documentar las responsabilidades y obligaciones éticas y profesionales, en un intento de educar y aleccionar a los ingenieros del software, a la vez que informar al público sobre las responsabilidades que son importantes para la profesión. Este código ético prima el bienestar y la calidad de vida del público en general, en cuanto a todas las decisiones relacionadas con la Ingeniería del Software [Gotterbarn et al., 1999b].

Los ingenieros informáticos deben responsabilizarse que al llevar a cabo sus actividades lo hagan con beneficio y respeto a la profesión que ejercen. De acuerdo a sus compromisos con la salud, la seguridad y el bien público, los ingenieros informáticos deben seguir los siguientes ocho principios [Gotterbarn, 1999]:

1. **Sociedad:** Los ingenieros del software actuarán de manera coherente con el interés general.
2. **Cliente y empresario:** Los ingenieros del software deberán actuar de tal modo que se sirvan los mejores intereses para sus clientes y empresarios, y consecuentemente con el interés general.
3. **Producto:** Los ingenieros del software deberán garantizar que sus productos y las modificaciones relacionadas con ellos cumplen los estándares profesionales de mayor nivel que sea posible.
4. **Juicio:** Los ingenieros del software deberán mantener integridad e independencia en su valoración profesional.
5. **Gestión:** Los gestores y líderes en Ingeniería del Software suscribirán y promoverán un enfoque ético a la gestión del desarrollo y el mantenimiento del software.
6. **Profesión:** Los ingenieros del software deberán progresar en la integridad y la reputación de la profesión, de forma coherente con el interés público.
7. **Compañeros:** Los ingenieros del software serán justos y apoyarán a sus compañeros.
8. **Persona:** Los ingenieros del software deberán participar en el aprendizaje continuo de la práctica de su profesión y promoverán un enfoque ético en ella.

Cuadro 4.1. Resumen del código ético de ACM/IEEE-CS, versión 5.2

En este capítulo se va a presentar qué[15] puede aportar a los futuros titulados las actividades a realizar desde la plaza a concurso, y que vienen marcadas por su perfil: *docencia en materias de Ingeniería del Software y Orientación a Objetos*. Para lo que primero se van a identificar las asignaturas relacionadas con el perfil de la plaza dentro del plan de estudios vigente, para presentar sus características, interrelaciones con otras

[15] El cómo se detalla en el capítulo 5 dedicado a la programación docente de las asignaturas.

asignaturas del plan de estudios y su situación tanto en el ámbito nacional como internacional.

4.1 El perfil de formación

Las actividades a realizar por la persona que ocupe la plaza a concurso se centran en impartir docencia en materias relacionadas con las disciplinas de Ingeniería del Software y de Tecnología de Objetos[16] en la *Ingeniería Técnica en Informática de Sistemas* dentro de la Universidad de Salamanca.

El siguiente paso es identificar qué asignaturas se ajustan al perfil de formación dentro del plan de estudios vigente en dicha titulación (*Plan de 1997, aprobado en el BOE de 4 de Noviembre de 1997*). En el *Capítulo 2* de este Proyecto Docente ya se presentó este plan de estudios, y más concretamente en la Tabla 2.14 se incluía una relación de las asignaturas troncales y obligatorias del mismo, mientras que en la Tabla 2.15 se hacía lo propio con las asignaturas optativas. Una descripción más detallada del plan de estudios vigente para la titulación de *Ingeniería Técnica en Informática de Sistemas* se encuentra en la *Guía Académica de la Facultad de Ciencias* para el Curso 1999-2000 [USAL, 1999].

De las asignaturas que se incluyen en este plan de estudios hay dos que se ajustan perfectamente al perfil propuesto para la plaza a concurso: ***Ingeniería del Software*** y ***Programación Orientada a Objetos***. Ambas asignaturas se imparten en el tercer curso de la *Ingeniería Técnica*, en el quinto y sexto cuatrimestre respectivamente.

La asignatura de *Ingeniería del Software* es una asignatura obligatoria de 6 créditos, 4,5 de ellos teóricos y el 1,5 restantes prácticos. Por su parte la asignatura de *Programación Orientada a Objetos* es una asignatura optativa de 6 créditos repartidos al cincuenta por ciento entre teoría y práctica.

Dado que el plan de estudios vigente comenzó su andadura en el curso 1997-1998, en el presente curso académico (1999-2000) es la primera vez que se cursan estas asignaturas, si bien ambas tenían asignaturas equivalentes, con las mismas características de nombre, obligatoriedad y carga docente en el plan de estudios anterior (Plan de 1992), como se puede comprobar en [USAL, 1998], apareciendo éstas como asignaturas sin docencia en el presente curso académico para aquellos alumnos que, matriculados en el Plan de 1992, no hubieran superado alguna de ellas, y quedando establecido el mecanismo de convalidación oportuno entre estas asignaturas del Plan de 1992 y sus homónimas del Plan de 1997.

[16] Nótese que a la Tecnología de Objetos se la puede considerar como un paradigma concreto de desarrollo, y por tanto como una parte de la Ingeniería del Software.

4.2 Ingeniería del Software

4.2.1 Introducción

Previamente al desarrollo de la propuesta concreta del programa de la asignatura *Ingeniería del Software*, que se desarrolla en el *Capítulo 5*, se presenta una visión global de la Ingeniería del Software como disciplina mediante un análisis de su marco histórico y conceptual. El objetivo no es realizar una exposición exhaustiva, sino mostrar aquellos conceptos, aportaciones y resultados que se consideran relevantes para el contenido de la asignatura objeto de la propuesta docente. De esta forma los contenidos se adaptarán a las tendencias actuales consolidadas en la teoría, tecnología, práctica y aplicación del software a los sistemas basados en computadores.

El desarrollo de un proyecto docente para una determinada disciplina implica, en primer lugar, la identificación clara y precisa del conjunto de conceptos y conocimientos científico/técnicos que la misma engloba. En el caso de la Ingeniería del Software es una tarea que presenta algunas dificultades debido al amplio conjunto de materias que abarca y a la constante evolución a la que se ve sometida como consecuencia de los rápidos cambios que sufre la tecnología del software. Una primera aproximación a esta tarea puede venir dada por la definición de Ingeniería del Software.

Antes de hacer un repaso por algunas de las muchas definiciones que de Ingeniería del Software se han dado, se va comentar el origen y polémica que el propio término ha suscitado.

La introducción del término Ingeniería del Software se produjo en la primera conferencia sobre Ingeniería del Software patrocinada por la OTAN, celebrada en Garmisch (Alemania) en octubre de 1968 [Naur and Randell, 1969], atribuyéndose la paternidad del término a **Fritz Bauer** [Randell, 1998].

Fue tal la aceptación de esta conferencia que se consiguió un nuevo patrocinio de la OTAN para una segunda conferencia sobre Ingeniería del Software, que tendría lugar un año más tarde en Roma (Italia) [Buxton and Randell, 1970], con unos resultados menos esperanzadores que los producidos en la primera conferencia. De hecho, no se produjo ninguna petición de que continuara la serie de conferencias de la OTAN, lo cual no influyó para que a partir de entonces se utilizara con gran profusión el nuevo término para describir los trabajos realizados, aunque quizás sin un consenso real sobre su significado.

En el contexto educativo, sin duda alguna, lo que más controversia ha levantado es el propio nombre del término, centrándose la discusión en la pregunta *¿es la Ingeniería del Software realmente una Ingeniería?* [Tomayko, 2000].

Los argumentos que se dan para sustentar una respuesta negativa se pueden resumir en dos categorías. La primera reuniría a aquéllos que se ciñen a la definición literal de Ingeniería dada por algunos diccionarios o sociedades profesionales, que se centran en que en estas definiciones se hace mención a productos tangibles derivados del uso efectivo de materiales y fuerzas naturales, mientras que el software ni es tangible, ni utiliza materiales y fuerzas naturales para su concepción.

La segunda categoría estaría formada por los que arguyen que una disciplina ingenieril evoluciona desde una profesión, y la profesión relacionada con el software no ha evolucionado lo suficiente para ser considerada una Ingeniería.

Por el contrario, son muchos los que están a favor de la utilización y difusión del término *Ingeniería del Software*, tomando como un estándar de facto la utilización reiterada del término en la bibliografía especializada.

Quizás la defensa más fuerte y adecuada de la Ingeniería del Software como Ingeniería venga de la mano de **Mary Shaw** que justifica que si tradicionalmente se ha definido Ingeniería como "*la creación de soluciones rentables a problemas prácticos mediante la aplicación del conocimiento científico para la construcción de cosas al servicio de la humanidad*" [Shaw, 1990], entonces el desarrollo del software es un problema ingenieril apropiado, porque involucra "*la creación de soluciones rentables económicamente para problemas prácticos*" [Shaw and Tomayko, 1991].

Una vez hechas estas disquisiciones sobre el término y sus controversias, se va a proceder a exponer una muestra de las numerosas definiciones que de Ingeniería del Software se pueden encontrar en la bibliografía:

> **"Ingeniería del software es el establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en máquinas reales"**
>
> **Fritz Bauer,** *First NATO Software Engineering Conference*, Garmisch (Germany), 1968; en **[Buxton et al., 1976]**[17]

> **"La aproximación sistemática al desarrollo, operación, mantenimiento y retirada del software"**
>
> **IEEE "Standard Glossary of Software Engineering Terminology" [IEEE, 1983]**

> **"Es la disciplina tecnológica y de gestión que concierne a la producción y mantenimiento sistemático de productos software que son desarrollados y modificados a tiempo y dentro de los costes estimados"**
>
> **[Fairley, 1985]**

[17] También en [Bauer, 1972].

"Tratamiento sistemático de todas las fases del ciclo de vida del software. Se refiere a la aplicación de metodologías para el desarrollo del sistema software"

Asociación Española para la Calidad.
"*Glosario de Términos de Calidad e Ingeniería del Software*" **[AECC, 1986]**

"Construcción de software multi-versión por un equipo de varias personas"

[Parnas and Weiss, 1987]

"La aplicación disciplinada de principios, métodos y herramientas de ingeniería, ciencia y matemáticas para la producción económica de software de calidad"

[Humphrey, 1989]

"La utilización de metodologías, herramientas y técnicas para resolver los problemas prácticos que surgen en la construcción, desarrollo, soporte y evolución del software"

Institute for Information Technology, NRC Canada, 1990

"Una disciplina cuyo objetivo es la producción de software de calidad, que se entrega en plazo, se ajusta al presupuesto y que satisface sus requisitos"

Vanderbilt University, "*Software Engineering*"
Aksen Assoc., 1990

"La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento del software; es decir, la aplicación de la ingeniería al software"

"*Standard Glossary of Software Engineering Terminology*". IEEE Std 610.12-1990
[IEEE, 1999]

"Disciplina tecnológica y de gestión concerniente a la invención, producción sistemática y mantenimiento de productos software de alta calidad, desarrollados a tiempo y al mínimo coste"

[Frakes et al., 1991]

"Las actividades sistemáticas implicadas en el diseño, implantación y prueba de software para optimizar su producción y soporte"

Canadian Standards Association.
"*CSA Information Technology Vocabulary*"**, 1992**

"Aplicación de herramientas, métodos y disciplinas para producir y mantener una solución automatizada de un problema real"

[Blum, 1992]

> **"Aquella forma de ingeniería que aplica principios de informática y matemática a la resolución de problemas software de forma eficiente en cuanto al coste"**
>
> **[Humphrey, 1993]**

> **"Aplicación de principios científicos para la transformación ordenada de un problema en una solución software funcional, así como en el consiguiente mantenimiento del software hasta el final de su vida útil"**
>
> **[Davis, 1993]**

> **"Es la aplicación de herramientas, métodos y disciplinas de forma eficiente en cuanto al coste, para producir y mantener una solución a un problema de procesamiento real automatizado parcial o totalmente por el software"**
>
> **[Horan, 1995]**

> **"La aplicación de métodos y conocimiento científico para crear soluciones prácticas y rentables para el diseño, construcción, operación y mantenimiento del software y los productos asociados, al servicio de las personas"**
>
> Adaptado de la definición de Ingeniería de
> **Mary Shaw y David Garlan** en **[Shaw and Garlan, 1996]**

En lugar de con definiciones escuetas, el *Instituto de Ingeniería del Software* y la *Sociedad Británica de Informática* presentan su visión de lo qué es la Ingeniería del Software con descripciones más elaboradas.

> 1. Definición central:
>
> - Ingeniería es la aplicación sistemática de conocimiento científico para la creación y construcción de soluciones rentables a problemas prácticos al servicio de la humanidad.
> - La Ingeniería del Software es la forma de ingeniería que aplica principios propios de la Ciencia de la Informática y Matemáticas para conseguir soluciones rentables a problemas software.
>
> 2. Elaboraciones e interpretaciones:
>
> - La creación y construcción del software debe incluir el mantenimiento. Debe cubrirse el ciclo de vida del software completo.
> - La rentabilidad implica no sólo dinero, sino tiempo, calendario y recursos humanos. También implica obtener buenos valores por los recursos invertidos; incluyendo la calidad cuando las medidas se consideren oportunas.
> - La Ingeniería del Software no se limita a aplicar sólo principios de la Ciencia de la Informática y las Matemáticas, sino cualquier principio del que pueda sacar ventaja.
> - La Ingeniería del Software necesita contar con principios y técnicas de gestión para llevar a cabo sus actividades de desarrollo.
>
> 3. Distinción entre el uso actual del término "Ingeniería del Software" y la definición que se adecua a la misión del SEI:
>
> - Actualmente, el término "Ingeniería del Software" tiene múltiples conjuntos de significados conflictivos y pobremente entendidos, que van desde la programación a la gestión del diseño del sistema.
> - Actualmente, el término "Ingeniería del Software" es más una aspiración que una descripción.
>
> **Software Engineering Institute (SEI) [Ford, 1990]**

"La Ingeniería del Software requiere la comprensión y aplicación de principios de ingeniería, habilidades de diseño, buenas prácticas de gestión, fundamentos de la Ciencia de la Informática y formalismos matemáticos. Es tarea de la Ingeniería del Software juntar estas áreas de trabajo tan dispares y utilizarlas en las fases de obtención de los requisitos, especificación, diseño, verificación, implementación, prueba, documentación y mantenimiento de sistemas software complejos y de gran tamaño. El ingeniero del software debe cumplir el papel del arquitecto del sistema complejo, tomando en cuenta las necesidades y requisitos del usuario, la viabilidad, el coste, la calidad, la confianza, la seguridad y las restricciones temporales. La necesidad de ajustar la importancia relativa de estos factores de acuerdo a la naturaleza del sistema y de su aplicación confiere una fuerte dimensión ética a las tareas del ingeniero del software, sobre quien puede depender la seguridad y bienestar de otros, y para quien, como en medicina o en derecho, se requiere un sentido de moralidad profesional para su trabajo.

El ingeniero del software debe ser capaz de estimar el coste y la duración del proceso de desarrollo del software, así como determinar la consecución de corrección y confianza. Tales medidas y estimaciones pueden involucrar conocimientos de conceptos financieros y de gestión, al mismo nivel que el manejo de los fundamentos matemáticos. Se necesita el uso preciso de las notaciones formales y de las palabras para expresarlas con el grado de precisión requerido a otros ingenieros y a clientes formados. En la mayoría de las circunstancias las hebras técnicas, teóricas y de gestión de un proyecto de Ingeniería del Software no pueden separarse unas de las otras.

Para construir grandes productos y conseguir una alta productividad, el ingeniero requiere el uso de herramientas software de desarrollo y de elementos reutilizables que garanticen su subsiguiente modificación y mantenimiento con seguridad.

La actividad profesional del ingeniero del software abarca el rango de tareas involucradas en el ciclo de vida de un sistema software. La obtención de requisitos, especificación, diseño, verificación y construcción son tareas críticas para conseguir la calidad del producto y son todas ellas responsabilidad del ingeniero del software.

Dado que el software determina el comportamiento de un autómata, el ingeniero del software necesita tener conocimientos de hardware digital y de comunicaciones. Aunque la Ingeniería del Software como disciplina puede ser calificada como independiente del área de aplicación, su realización debe ser en el contexto de aplicaciones específicas. El ingeniero del software debe, por tanto, ser capaz de colaborar con otros profesionales que le brindarán capacidades complementarias en la labor de especificar, diseñar y construir sistemas hardware-software que se ajusten a las necesidades del cliente, haga uso de las soluciones hardware y software en una óptima combinación y ofrezca una interfaz de usuario con una calidad adecuada.

La mayoría del software se construye en equipo, frecuentemente con equipos interdisciplinarios. La habilidad para trabajar cerca los unos de los otros es esencial.

Algunos de los métodos y de las herramientas intelectuales de la Ingeniería del Software están en proceso de desarrollo y se espera que tengan que cambiar de forma rápida. Los ingenieros del software, por tanto, necesitan tener unos buenos fundamentos teóricos que les sirvan de base par aprender y usar nuevos métodos en el futuro, y la mentalidad que les permita actualizar de forma permanente los conocimientos que necesitan para su labor profesional"

British Computer Society and the Institution of Electrical Engineers [BCS, 1989]

No sólo hace falta decir lo qué es la Ingeniería del Software, sino que es conveniente recalcar lo qué **no** es; así la Ingeniería del Software no es el diseño de programas que se implementan en otras áreas ingenieriles, ni es simplemente una forma de programar más organizada que la que prevalece entre aficionados, principiantes o personas con falta de educación y entrenamiento específico.

Esta variedad de definiciones refleja las diferentes concepciones existentes sobre la Ingeniería del Software. A modo de ejemplo, esta diferencia de alcance y concepción de la Ingeniería del Software como disciplina se aprecia revisando los contenidos del libro de **Ian Sommerville** [Sommerville, 1985], donde la Ingeniería del Software está limitada al desarrollo de la programación mientras que el libro de **Roger S. Pressman** [Pressman, 1987] de esa misma época ya considera el análisis y diseño de sistemas. En su última edición hasta la fecha Sommerville [Sommerville, 1996] amplía su concepto de Ingeniería del Software a aquellas actividades relacionadas con la especificación, desarrollo, gestión y evolución del software, no incorporando ningún capítulo específico sobre la práctica o los lenguajes de programación.

El concepto de Ingeniería del Software surge de la distinción entre programación de pequeños proyectos (*programming in the small*) y programación de grandes proyectos (*programming in the large*) y el reconocimiento de que la Ingeniería del Software está

relacionada con esta última. Este primer concepto fue rápidamente ampliado para incorporar a la Ingeniería del Software todas aquellas tareas relacionadas con la automatización de los Sistemas de Información y con la Ingeniería de Sistemas en general.

El enfoque de la Ingeniería del Software que se adopta en este proyecto es el relacionado con los problemas que se presentan en el desarrollo de grandes sistemas software bajo la perspectiva de los sistemas de información a los que dan servicio. Esto viene motivado por el hecho de que aquellos enfoques de la Ingeniería del Software más relacionados con el diseño y la programación de módulos o componentes software están asignados a otras asignaturas del plan de estudios como *Programación, Laboratorio de Programación, Estructuras de Datos* o *Programación Orientada a Objetos*.

4.2.2 Marco histórico de la Ingeniería del Software

El contexto en el cual se han desarrollado las técnicas del software está íntimamente ligado a la evolución solapada de los sistemas informáticos y de la programación, cuyos hitos o avances más importantes han configurado las eras o etapas que se detallan a continuación. No se intenta hacer una clasificación cronológica exacta ya que los desarrollos han estado muy solapados e incluso ideas que surgieron en una fecha determinada no serian aceptadas ampliamente quizá hasta 10 años después [Goldberg, 1986], [Pressman, 1992].

Ninguna de estas eras puede decirse que hayan terminado formalmente, sobre todo si se hace caso del estudio [Redwine, 1985] que afirma que el tiempo necesario para aceptar una idea que implique un cambio tecnológico es de 15 a 20 años. Esta afirmación ha sido corroborada más recientemente por otros muchos autores, como **Klaus Dittrich** del grupo del conocido "***manifiesto de Atkinson***" [Atkinson et al., 1989].

Durante la **primera era** (1.950-1.96x), la programación de ordenadores se concibe más como un arte que como una técnica sistematizada y compleja. En esta época la importancia se centra en el hardware, sometido a un cambio continuo, considerando al software como un *añadido*. El software se desarrolla utilizando únicamente la intuición, con escasos métodos sistematizados y prácticamente sin ningún control en su desarrollo. La mayoría de los sistemas utilizaban una orientación *batch* (*excepciones a esto son el sistema de reserva de American Airlines y los sistemas americanos de defensa en tiempo real, como SAGE*). El ordenador estaba dedicado a la ejecución de un programa simple que a su vez resolvía una situación específica.

En la **segunda era** (1.96x-197x) la multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos en la interacción hombre-máquina. Los sistemas en tiempo real podían recoger, analizar y transformar datos procedentes de múltiples orígenes, controlando procesos y produciendo resultados en milisegundos en vez de en

minutos. Los avances en las máquinas de memoria secundaria *on-line* llevaron a la primera generación de sistemas de gestión de bases de datos.

La segunda era también se caracterizó por el uso de *productos software* o paquetes de software y por el comienzo de las "*software house*" o "*casas de servicios*". El software empezó a ser desarrollado pensando en una distribución más amplia y para un mercado multidisciplinar.

Una de las primeras contribuciones a la ingeniería del software, en el sentido de mejorar la fiabilidad, dirección y productividad en el desarrollo del software, fue el artículo de **E. W. Dijkstra** "*Go to Statement Considered Harmful*", que apareció en la revista **Communications of the ACM** (Vol. 11, N.3) en 1968, y acentuaba los beneficios de utilizar los conceptos de la programación estructurada. Otra contribución importante de esta época fue la obra de **Weinberg** "*The Psychology of Computer Programming*" [Weinberg, 1971]; este clásico de la literatura del software introdujo las ideas de "*programación sin ego*", nombres mnemónicos de variables y en general la necesidad de la claridad y un buen estilo en la programación.

Según fue creciendo el número de sistemas basados en ordenador las bibliotecas de software se fueron extendiendo. Los proyectos produjeron cientos de miles de instrucciones fuente. Pero los problemas también empezaron a aparecer: todos estos sistemas informáticos y todas estas instrucciones fuente tenían que ser mantenidos para corregir errores "*oscuros*" (detectados tardíamente), adaptarse a los cambios en los requisitos de los usuarios o adaptarse al nuevo hardware que adquirían las organizaciones. El esfuerzo necesario para el mantenimiento de los sistemas informáticos, y sobre todo del software asociado, comenzó a absorber recursos de forma alarmante y, peor aún, la naturaleza personalizada y la ausencia o escasez de técnicas generales de diseño y análisis, hacía que muchos de estos sistemas fuesen prácticamente inmantenibles: había empezado una **crisis del software**, reconocida por primera vez oficialmente en la reunión de la **OTAN** de 1968 [Naur and Randell, 1969], [Boehm, 1976], [Goldberg, 1986], [Randell, 1998].

En la **tercera era** (1.97x-1.98x) aparecen los sistemas distribuidos - varios computadores, realizando cada uno sus funciones concurrentemente y comunicándose unos con otros - dando lugar a un incremento importante de la complejidad de los sistemas informáticos. Esta época se caracteriza por el uso generalizado del microprocesador que, gracias al abaratamiento y aumento de potencia de éstos, permite la realización de funciones complejas a un coste excepcionalmente bajo.

Durante esta era, la crisis del software se acentuó, detectándose que aproximadamente el **50%** de los presupuestos de los centros de procesamiento de datos se dedica a mantenimiento [Goldberg, 1986], con lo que la productividad en el desarrollo de nuevos sistemas se vio notablemente dañada. La intensificación de la crisis provoca como respuesta una mayor toma en consideración de la necesidad de un proceso de Ingeniería para el desarrollo del software. Como consecuencia de la

importancia que adquiere la Ingeniería del Software hacen su aparición las primeras metodologías, destacando las propuestas por Jackson [Jackson, 1975], Warnier [Warnier, 1974] y DeMarco [DeMarco, 1979] por ser las que mayor difusión y utilización alcanzan. Por otra parte, este acercamiento del proceso de construcción del software a los procesos de ingeniería clásicos, conduce a la aplicación de técnicas de gestión de proyectos como PERT y CPM a los proyectos de desarrollo de software.

La consecuencia que se extrae de esta época es que es mejor utilizar alguna metodología disciplinada, no importa cual, que no utilizar ninguna [Basili, 1991].

Una **cuarta era** (1.98x-...) ha venido caracterizada por la introducción de sistemas de sobremesa, y la adopción de tecnologías y herramientas que proporcionan el soporte necesario para mejorar la calidad y la productividad en el desarrollo del software. Entre ellas se pueden destacar: *las herramientas CASE, los entornos de programación, el prototipado rápido (usado independientemente o con el ciclo de vida tradicional)*, *la tecnología orientada a objetos, la reutilización sistemática del software, y los lenguajes de cuarta generación (4GL)*.

Es también en este período cuando empiezan a darse a conocer las aproximaciones formales al desarrollo de software a través de especificaciones algebraicas y lenguajes de especificación ejecutables [Goguen and Meseguer, 1988], como OBJ [Goguen et al., 1992] o ACT ONE [Ehrig and Mahr, 1985], técnicas y métodos orientados a modelos como Z [Spivey, 1989], [Diller, 1990] o VDM [Jones, 1990] y álgebras de procesos como CSP [Hoare, 1985], aunque algunas de estas técnicas son muy anteriores (*como VDM, que fue desarrollado por los laboratorios de investigación de IBM de Viena, en 1973*).

En los últimos años, han cobrado una gran importancia los modelos de proceso y la preocupación por la mejora en la calidad tanto del producto software como del proceso para mejorarlo. De estos modelos se puede destacar la norma ISO-9000, en su aplicación a la construcción de software [Layman, 1994], [Schmauch, 1994], la iniciativa "BOOTSTRAP" [Lebsanft and Synspace, 1994], que consiste en un método para analizar y evaluar cuantitativamente determinados atributos de calidad del proceso (la evaluación permite conseguir un perfil detallado acerca de la calidad de la organización donde se aplica); el conocido modelo de madurez y capacidad de la Universidad Carnegie-Mellon (Pittsburg, PA - USA), CMM (*Capability Maturity Model*) [Paulk et al., 1993a], [Paulk et al., 1993b] o el método SPICE (*Software Process Improvement & Capability Evaluation*) [Dorling, 1993], [Konrad et al., 1995] que tiene como objetivo convertirse en un estándar ISO mundial que integre a las iniciativas anteriores.

Los sistemas basados en microprocesadores de 32 ó 64 bits, la computación paralela, el desarrollo de sistemas de Inteligencia Artificial (*todavía hoy no excesivamente extendidos en el mercado*) y las nuevas tecnologías (*láser, fibra óptica...*) en las comunicaciones (*Internet*), junto con herramientas de programación visual,

desarrollo y ejecución de aplicaciones remotas, están condiciendo a **la transición hacia una quinta era**. En ingeniería del software se espera que esta nueva era venga marcada por el desarrollo y perfeccionamiento de los entornos de programación y herramientas integradas de apoyo a metodologías, por la continuación y mejora de las técnicas asociadas al prototipado y reusabilidad del software y por la aplicación de técnicas de "Ingeniería del Conocimiento" al desarrollo de software, especialmente como apoyo a la Ingeniería de Requisitos en la captura, estructuración y reutilización del conocimiento (*requisitos*) en dominios de aplicación [Sutcliffe and Maiden, 1998].

4.2.3 Marco conceptual de la Ingeniería del Software

En la introducción se señaló que el enfoque de la Ingeniería del Software que aquí se presenta es el relacionado con los Sistemas de Información a los que da servicio. Bajo esta perspectiva, un sistema software es una parte de un sistema mayor que lo engloba como componente. La Ingeniería del Software, por lo tanto, será solamente una parte del diseño del sistema en la que los requisitos del software han de ajustarse a los requisitos del resto de los elementos que constituyen este sistema. Por este motivo, el ingeniero del software ha de estar implicado en el desarrollo de los requisitos del sistema completo, comprendiendo el dominio de actividad en su totalidad.

Atendiendo a esta concepción de la Ingeniería del Software, es importante remarcar el papel que desempeña la Teoría General de Sistemas como antecedente conceptual en el que se apoya la teoría sobre los Sistemas de Información a los que la Ingeniería del Software intenta aportar soluciones. En el marco de la Teoría General de Sistemas, el análisis de sistemas tiene como objetivo general la comprensión de los sistemas complejos para abordar su modificación de forma que se mejore el funcionamiento interno para hacerlo más eficiente, para modificar sus metas... Las modificaciones pueden consistir en el desarrollo de un subsistema nuevo, en la agregación de nuevos componentes, en la incorporación de nuevas transformaciones... En general, el análisis de sistemas establece los siguientes pasos a seguir:

1. ***Definición del problema***. *En este paso se identifican los elementos de insatisfacción, los posibles cambios en las entradas y/o salidas al sistema y los objetivos del análisis del sistema.*
2. ***Comprensión y definición del sistema***. *En este paso se identifica y descompone el sistema jerárquicamente en sus partes constituyentes o subsistemas junto con las relaciones existentes entre los mismos.*
3. ***Elaboración de alternativas***. *En este paso se estudian las diferentes alternativas existentes para la modificación y mejora del sistema, atendiendo a los costes y perspectivas de realización.*
4. ***Elección de una de las alternativas definidas en el paso anterior.***
5. ***Puesta en práctica de la solución elegida.***
6. ***Evaluación del impacto de los cambios introducidos en el sistema.***

Muchas de las técnicas y métodos actuales de la Ingeniería del Software intentan dar respuesta a este tipo de cuestiones.

Los sistemas, de los que el software forma parte, se denominan sistemas informáticos o sistemas de computadora. En este sentido **Roger S. Pressman** [Pressman, 1997] considera que la Ingeniería del Software ocurre como consecuencia de un proceso denominado Ingeniería de Sistemas de Computadora. La Ingeniería de Sistemas de Computadora se concentra en el análisis, diseño y organización de los elementos en un sistema que pueden ser un producto, un servicio o una tecnología para la transformación de información o el control de información. De igual forma, este autor denomina al *Proceso de Ingeniería del Software* como **Ingeniería de la Información** cuando el contexto de trabajo de Ingeniería se enfoca a una empresa, y lo denomina **Ingeniería de Producto** cuando el objetivo es construir un producto. El término genérico de Ingeniería de Sistemas es el que utiliza para unificar estos dos tipos de ingenierías. La Ingeniería de Sistemas establece, por lo tanto, el papel que ha de asignarse al software y los enlaces que unen al software con otros elementos de un sistema basado en computadora.

Desde una perspectiva más general, **J. L. Le Moigne** [Le Moigne, 1973] concibe los sistemas formados por tres subsistemas interrelacionados: *el de decisión, el de información* y *el físico*. El sistema de decisión procede a la regulación y control del sistema físico decidiendo su comportamiento en función de los objetivos marcados. El sistema físico transforma un flujo físico de entradas en un flujo físico de salidas. En interconexión entre el sistema físico y el sistema de gestión se encuentra el sistema de información. El sistema de información está compuesto por diversos elementos encargados de almacenar y tratar las informaciones relativas al sistema físico a fin de ponerlas a disposición del sistema de gestión. El Sistema Automatizado de Información (SAI) es un subsistema del sistema de información en el que todas las transformaciones significativas de información son efectuadas por máquinas de tratamiento automático de las informaciones.

Basándose en las ideas anteriores, se considera a la Ingeniería del Software como la disciplina que se ocupa de las actividades relacionadas con los sistemas informáticos o sistemas de información en los que el software desempeña un papel relevante. Estos sistemas de información han de ser fiables, es decir, que su realización se lleve a cabo de forma correcta conforme a unos estándares de calidad y, además, que su desarrollo se realice en el tiempo y coste establecidos. Este último aspecto es crucial, y existe una gran variedad de informes, publicaciones y datos que avalan la gran dependencia que las organizaciones tienen hoy en día de los sistemas software.

En muchas ocasiones la Ingeniería del Software se ha querido limitar al desarrollo de grandes proyectos informáticos. Sin embargo, tal y como afirma **Barry B. Boehm** "*se hacen planos para una casa tanto si esta es grande como si es pequeña*". La filosofía que subyace en esta frase, es que cualquier desarrollo de software ha de seguir

un proceso. Como afirma **Roger S. Pressman** [Pressman, 1997]: "*El fundamento de la Ingeniería del Software es la capa proceso. El proceso de la Ingeniería del Software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la Ingeniería del Software*". El proceso define un marco de trabajo en el que se establece el control de gestión de los proyectos software y el contexto en el que se aplican los métodos técnicos y se producen los resultados del trabajo.

En el proceso de construcción de sistemas informáticos se pueden distinguir dos fases genéricas, independientemente del paradigma de ingeniería elegido: ***la definición*** y ***el desarrollo***.

Durante la fase de *definición* se identifican los requisitos claves del sistema y del software. Durante la misma se desarrollan un **Análisis de Sistemas**, en el que se define el papel de cada elemento en el sistema automatizado de información, incluyendo el que jugará el software, y un **Análisis de Requisitos** en el que se especifican todos los requisitos de usuario que el sistema tiene que satisfacer. Esta fase está orientada al **QUÉ**: *qué información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, qué ligaduras de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto*. En la fase de definición existe un paso complementario que consiste en la planificación del proyecto software, en el que se asignan los recursos, se estiman los costes y se planifican las tareas y el trabajo.

Por el contrario la fase de *desarrollo* está orientada al **CÓMO**, y el primer paso de esta fase corresponde al **Diseño del Software**. En el diseño del software *se trasladan los requisitos del software a un conjunto de representaciones que describen la estructura de datos, arquitectura del software y procedimientos algorítmicos y que permiten la construcción física de dicho software*. Los otros dos pasos de la fase de desarrollo corresponden a la **Codificación** y a la **Prueba del Software**.

Además de las fases de definición y desarrollo del software, existe una tercera fase dentro de la construcción de los sistemas que corresponde a la fase de **Mantenimiento** de los mismos. Esta fase se enfoca *a los cambios asociados con la corrección de errores, con las adaptaciones requeridas por la evolución del entorno del software y las modificaciones debidas a los cambios de requisitos del usuario para mejorar el sistema*.

La fase de definición es crucial en el desarrollo de un sistema software pues en ella quedan establecidas y determinadas explícitamente las necesidades y limitaciones del usuario y del sistema. La especificación de requisitos ha de realizarse antes de que la construcción del sistema de comienzo, pues de lo contrario podría ocurrir que las necesidades se simplifiquen completamente, las limitaciones se olviden o las interdependencias se pasen por alto durante la fase de desarrollo.

Para la comprensión y validación del sistema en estudio, se elaboran modelos. Estos modelos han de separar las especificaciones conceptuales y lógicas (¿qué ha de

cumplir el sistema?), de las especificaciones físicas, (¿cómo lo hará dependiendo de los recursos hardware y software?). Un concepto esencial en el desarrollo de este proceso es el de "***nivel de abstracción***". Aplicando este concepto, el desarrollo de un sistema de información consiste en definir una jerarquía apropiada de niveles de abstracción. Cada nivel produce un modelo del sistema que se describe mediante un lenguaje apropiado. El desarrollo comienza con niveles de abstracción altos, poco detallados, y termina con los de máximo detalle que sirven como base para la construcción directa del sistema ejecutable.

Una última consideración a tener en cuenta en el proceso de construcción de sistemas informáticos, es que la operatividad del producto final depende en gran medida de su conocimiento. Esto se consigue con la elaboración detallada y precisa de la documentación de apoyo: *documentación de sistema, manual de usuario, instrucciones de instalación, guías de entrenamiento, manual de operación...*

El uso de una **metodología** permite el dominio del proceso descrito, definición, desarrollo, implementación y mantenimiento, lo que asegurará el éxito de los proyectos informáticos. En general, una metodología es "*el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*" [DRAE, 1995]. Se puede decir que una metodología es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería del Software. A su vez, un método, es un procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla. Es un conjunto de técnicas, herramientas y tareas que, de acuerdo a un enfoque metodológico, se aplican para la resolución de un problema.

Desde el punto de vista específico de la Ingeniería del Software, la metodología describe como se organiza un proyecto, el orden en el que la mayoría de los trabajos tienen que realizarse y los enlaces entre ellos, indicando asimismo cómo tienen que realizarse algunos trabajos proporcionando las herramientas concretas e intelectuales. En concreto, se puede definir **metodología de Ingeniería del Software** como "*un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas*" [Rumbaugh et al, 1991].

En la actualidad se pueden distinguir seis escuelas principales de pensamiento en relación con las técnicas y métodos de desarrollo de Ingeniería del Software:

1. **Orientadas a procesos**: Si se parte de que la Ingeniería del Software se fundamenta en el modelo básico **entrada/proceso/salida** de un sistema[18]; de forma que los datos se introducen en el sistema y éste responde ante ellos transformándolos para obtener salidas. Estas metodologías se enfocan fundamentalmente en la parte de proceso y, por esto, se describen como un enfoque de desarrollo de software orientado al proceso. Utilizan un enfoque de descomposición descendente para evaluar los procesos del espacio del

[18] Este modelo básico es utilizado por todas las metodologías estructuradas.

problema y los flujos de datos con los que están conectados. Este tipo de metodologías se desarrolló a lo largo de los años 70. Los creadores de este tipo de métodos fueron **Edward Yourdon y Larry Constantine** [Yourdon and Constantine, 1975], [Yourdon and Constantine, 1979], [Yourdon and Constantine, 1989], [Yourdon, 1989]; **Tom DeMarco** [DeMarco, 1979]; **Gane y Sarson** [Gane and Sarson, 1977], [Gane and Sarson, 1979]. Representantes de éste grupo son las metodologías de análisis y diseño estructurado como **YSM** (Yourdon Systems Method) [Yourdon Inc., 1993], **SSADM** (Structured Systems Analysis and Design Method) [Ashworth and Goodland, 1990] o **METRICA v.2.1** [MAP, 1995].

2. **Orientadas a datos**: Al contrario que en el caso anterior, estas metodologías se centran más la parte de **entrada/salida** dentro del modelo básico **entrada/proceso/salida.** En estas metodologías las actividades de análisis comienzan evaluando en primer lugar los datos y sus interrelaciones para determinar la arquitectura de datos subyacente. Cuando esta arquitectura está definida, se definen las salidas a producir y los procesos y entradas necesarios para obtenerlas. Ejemplos representativos de este grupo son los métodos **JSP** (Jackson Structured Programming) y **JSD** (Jackson Structured Design) [Jackson, 1975], [Jackson, 1983], [Cameron, 1989], la construcción lógica de programas **LCP** (Logical Construction Program) de [Warnier, 1974] y el **DESD** (Desarrollo de Sistemas Estructurados de Datos), también conocido como metodología **Warnier-Orr**, [Orr, 1977], [Orr, 1981].

3. **Orientadas a estados y transiciones**: Estas metodologías están dirigidas a la especificación de sistemas en tiempo real y sistemas que tienen que reaccionar continuamente a estímulos internos y externos (eventos o sucesos). Las extensiones de las metodologías de análisis y diseño estructurado de **Ward y Mellor** [Ward and Mellor, 1985] y de **Hatley y Pirbhai** [Hatley and Pirbhai, 1987] son dos buenos ejemplos de estas metodologías.

4. **Diseño basado en el conocimiento**: Es una aproximación que se encuentra aún en una fase temprana de desarrollo. Utiliza técnicas y conceptos de Inteligencia Artificial para especificar y generar sistemas de información. El método **KADS** (Knowledge Acquisition and Development Systems) [Wielinga et al. 1991] y la metodologías **IDEAL** [Gómez et al., 1998] son ejemplos de esta categoría.

5. **Orientadas a objetos**: Estas metodologías se fundamentan en la integración de los dos aspectos de los sistemas de información: datos y procesos. En este paradigma un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes. El objeto encapsula datos y

operaciones. Este enfoque permite un modelado más natural del mundo real y facilita enormemente la reusabilidad. Algunos representantes de este grupo son las metodologías **OOA/D** de **Grady Booch** [Booch, 1994], **OMT** (Object Modeling Technique) [Rumbaugh et al., 1991], **OOSE** (Object Oriented Software Engineering) de [Jacobson et al., 1993], **FUSION** propuesta por [Coleman et al., 1994], **MOSES** de [Henderson-Sellers and Edwards, 1994a] o **RUP** (Rational Unified Process) [Jacobson et al., 1999].

6. **Basadas en métodos formales**: Estas metodologías implican una revolución en los procedimientos de desarrollo, ya que a diferencia de todas las anteriores, estas técnicas se basan en teorías matemáticas que permiten una verdadera aproximación científica y rigurosa al desarrollo de sistemas de información y software asociado.

Existen otras clasificaciones de las metodologías, por ejemplo en [Piattini et al., 1996] éstas se clasifican en función de tres parámetros *el enfoque, el tipo de sistema* y *la formalidad*.

Atendiendo a todo lo expuesto parece imprescindible incluir en la asignatura de Ingeniería del Software los conceptos relacionados con los actores implicados en el desarrollo de proyectos, aspectos del desarrollo y la calidad de los productos finales, así como los procedimientos, herramientas y métodos de trabajo a disposición del analista para poder construir el software de calidad que el usuario necesita. Teniendo en cuenta que entre los enfoques metodológicos mencionados anteriormente la Orientación a Objetos es una de las líneas más prometedoras y que las orientadas a procesos y las orientadas a estados y transiciones siguen siendo las más utilizadas y difundidas en la actualidad, estos serán los enfoques metodológicos que se incluirán en el programa de la asignatura.

Una vez que se ha introducido cual es el marco conceptual de la asignatura, se va a perfilar de una manera más concreta el conjunto de conocimientos que entrarían dentro del área de influencia de la Ingeniería del Software: *su cuerpo de conocimiento*.

4.2.3.1 El cuerpo de conocimiento de la Ingeniería del Software

Como se indicó al comienzo de este capítulo una de las tareas básicas para la definición de una profesión es el establecimiento del conjunto de conocimientos que el profesional debe poseer para el adecuado ejercicio de su labor profesional. Este cuerpo de conocimiento es fundamental para constituir el resto de los elementos que conformarán la profesión, esto es, una propuesta curricular y una política de certificación de los estudios y de los profesionales.

Ante esta necesidad se han propuesto diferentes alternativas, algunas de las cuales van a ser presentadas someramente en los siguientes apartados.

SWEBOK propuesto por IEEE-CS y ACM

De las diferentes propuestas el proyecto **SWEBOK** (*Software Engineering Body of Knowledge*), patrocinado por **IEEE-CS** y **ACM**, es el que acabará acatándose como estándar internacional, aunque a fecha de hoy todavía no se ha finalizado estando en la segunda fase (*versión del hombre de piedra* [Abran et al., 1999], [Abran et al., 2000]) de las tres fases de que consta el proyecto, esperando que finalice a lo largo del año 2001, como se aprecia en la Figura 4.1.

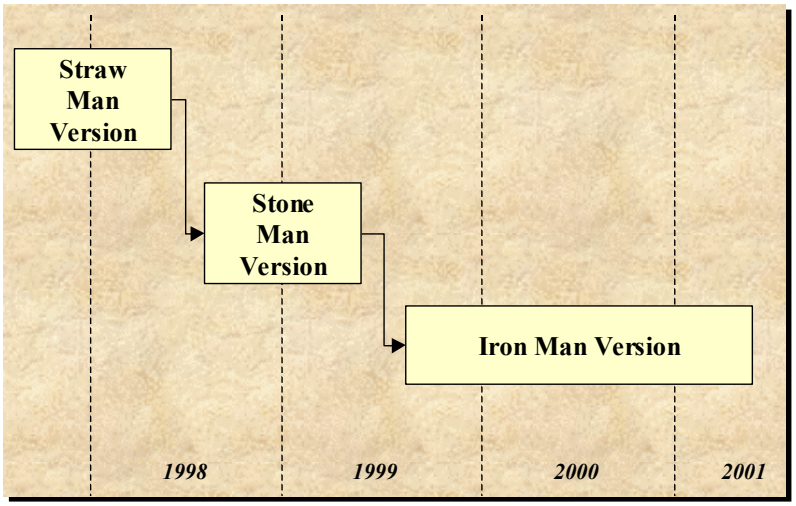

Figura 4.1. Duración aproximada del proyecto SWEBOK

Los objetivos de este proyecto se resumen en los siguientes cinco puntos [Bourque et al., 1999a]:

1. Caracterizar los contenidos de la Ingeniería del Software como disciplina.
2. Ofrecer un acceso al cuerpo de conocimiento de la Ingeniería del Software.
3. Promover una vista consistente de la Ingeniería del Software para todo el mundo.
4. Clarificar el lugar, y establecer los límites, de la Ingeniería del Software con respecto a otras disciplinas, tales como la Ciencia de la Informática, la Gestión de Proyectos, la Ingeniería de Computadores o las Matemáticas.
5. Ofrecer las bases para el desarrollo de una propuesta curricular y una política de certificación, relacionadas ambas con la Ingeniería del Software.

El producto que resulte de este proyecto no será el cuerpo de conocimiento en sí, sino más bien una guía de él. El conocimiento ya existe, lo que se busca es el consenso para determinar el subconjunto de conceptos esenciales que caracterizan a la Ingeniería del Software.

La guía que, como ya se ha mencionado, actualmente se encuentra en la parte final de su segunda fase (*versión del hombre de piedra*), se divide en diez áreas de conocimiento, con una serie de especialistas responsables de cada una de ellas, que se recogen en la Tabla 4.1.

Área de Conocimiento	Especialistas
Gestión de la configuración del software	J.A. Scott y D. Nisse (Lawrence Livermore Laboratory, USA)
Construcción del software	T. Bollinger (The Mitre Corporation, USA)
Diseño del software	G. Tremblay (Université du Québec à Montreal, Canadá)
Infraestructura de la Ingeniería del Software	D. Carrington (The University of Queensland, Australia)
Gestión de la Ingeniería del Software	S.G. MacDonell y A.R. Gray (University of Otago, Nueva Zelanda)
Proceso de Ingeniería del Software	K. El Emam (National Research Council, Canadá)
Evolución y mantenimiento del software	T.M. Pigoski (Techsoft, USA)
Análisis de la calidad del software	D. Wallace y L. Reeker (National Institute of Standards and Technology, USA)
Análisis de los requisitos del software	P. Sawyer y G. Kotonya (Lancaster University, Reino Unido)
Prueba del Software	A. Bertolino (National Research Council, Italia)

Tabla 4.1. Las áreas de conocimiento del SWEBOK y sus responsables

Además, se han considerado siete disciplinas relacionadas con la Ingeniería del Software: *Ciencias cognitivas y factores humanos, Ingeniería de computadores, Ciencia de la Informática, Gestión y ciencia de la gestión, Matemáticas, Gestión de proyectos* e *Ingeniería de Sistemas*.

El proyecto SWEBOK especifica las unidades de conocimiento, así como los temas pertenecientes a dichas áreas de conocimiento, que se considerará el conocimiento esencial que debe poseer un ingeniero del software. Éstos deben también poseer ciertos conocimientos de las disciplinas relacionadas, pero no es cometido del SWEBOK especificar estos conocimientos.

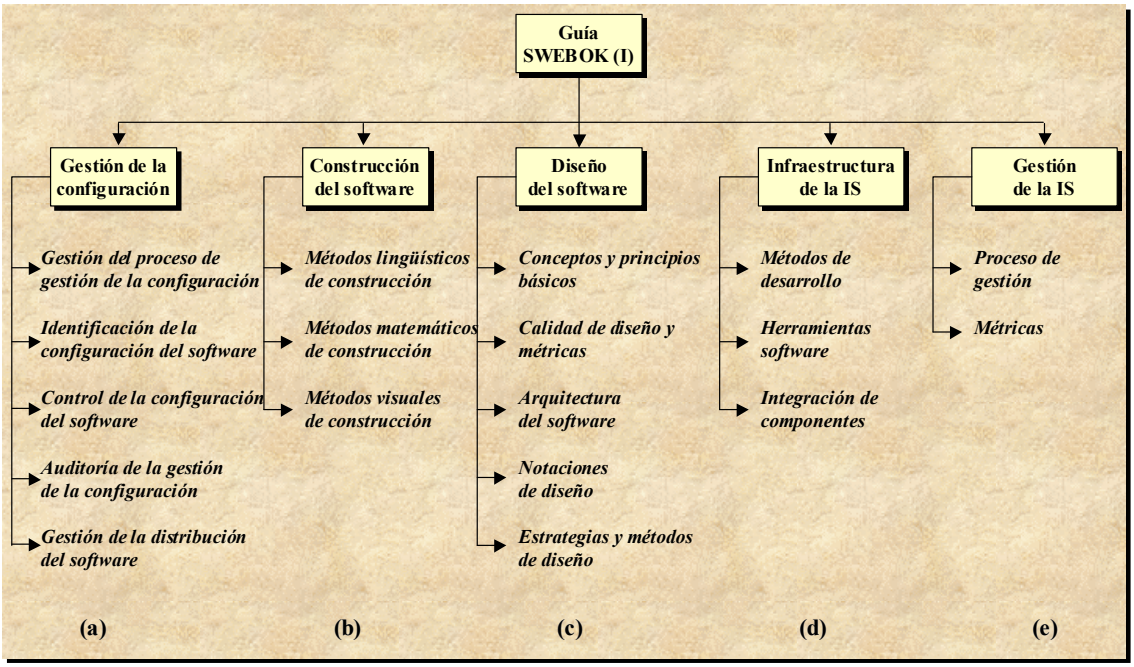

Figura 4.2. Estructura de la guía SWEBOK (parte I)

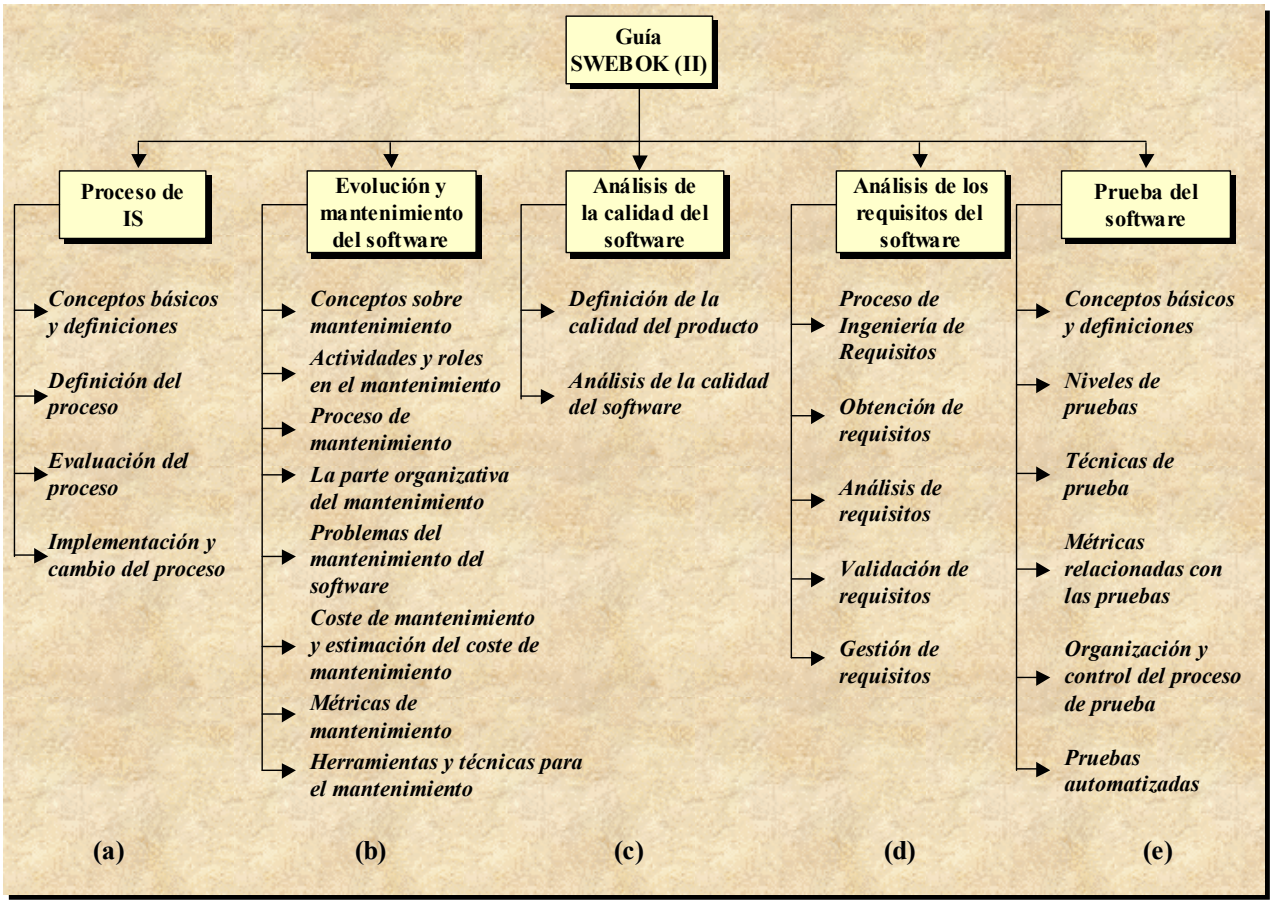

Figura 4.3. Estructura de la guía SWEBOK (parte II)

La guía del SWEBOK se organiza de forma jerárquica, descomponiendo cada área de conocimiento en un conjunto de temas con nombres fácilmente reconocibles, como se puede apreciar en la Figura 4.2 y en la Figura 4.3. Una breve descripción de cada una de las diez áreas de conocimiento se presenta en el Cuadro 4.2.

El número de áreas de conocimiento no ha sido siempre de diez, y muy probablemente este número variará en la versión definitiva de la guía. En la *Straw Man Version* [Bourque et al., 1998] fueron quince las áreas de conocimiento consideradas; éstas se obtuvieron del estudio de libros de texto sobre Ingeniería del Software, del estudio de programas de cursos universitarios y masters en Ingeniería del Software, y especialmente el ciclo de vida **ISO/IEC 12207**[19] [ISO/IEC, 1995] fue considerado como una entrada principal para esta versión de la guía, marcando las bases y el vocabulario para la clasificación de los diferentes temas relacionados con el ciclo de vida.

[19] Adoptado por IEEE/EIA y por ISO/IEC como un estándar.

Gestión de la configuración del software: Es la disciplina que identifica la configuración de puntos discretos en el tiempo para lograr un control sistemático de sus cambios y mantener la integridad y la trazabilidad a través del ciclo de vida del sistema.

Construcción del software: Es el acto fundamental de la Ingeniería del Software. Requiere el establecimiento de un diálogo entre el ingeniero y el computador, entre representantes de dos mundos totalmente diferentes. Para establecer los temas de esta unidad de conocimiento se adoptan dos vistas complementarias de la construcción del software. La primera de ellas establece las tres interfaces principales para la creación de software: *lingüista, matemática* y *visual*. La segunda establece que, para cada uno de los estilos, los temas a tratar se organicen de acuerdo a cuatro principios de organización: *reducción de la complejidad, anticipación de la diversidad, estructuración de la validación* y *uso de estándares externos*.

Diseño del software: Transforma los requisitos, típicamente expresados en términos del dominio del problema, en descripciones que explican cómo resolver el problema. Describe cómo el sistema se descompone y se organiza en componentes, describiendo las interfaces entre ellos.

Infraestructura de la Ingeniería del Software: Describe tres subáreas que discurren de forma horizontal a través de otras áreas de conocimiento: *los métodos de desarrollo, las herramientas software* y *la integración de componentes*.

Gestión de la Ingeniería del Software: Une la gestión del proceso y la parte de métricas del proceso. La gestión del proceso casa con la noción de "*gestión a la larga*", esto es, trata de la organización de las fases del ciclo de vida. La parte de métricas aborda *la medida de los objetivos del programa, la medida de la selección, la recolección de datos* y *el desarrollo de modelos*.

Proceso de la Ingeniería del Software: Cubre la definición, implementación, medida, gestión, cambio y mejora de los procesos software.

Evolución y mantenimiento del software: Estudia los procesos relacionados con la modificación de un producto software después de su entrega, para corregir faltas, mejorar su rendimiento u otros atributos, o adaptar el producto a otro entorno. Pero sin embargo, normalmente más que considerar que un sistema software está terminado, se piensa que evoluciona constantemente, de ahí la inclusión de la parcela de evolución del software.

Análisis de la calidad del software: Discute sobre el aseguramiento de la calidad que todo producto resultado de un proceso de Ingeniería del Software debe tener.

Análisis de los requisitos del software: Se divide en cinco subáreas que se corresponden aproximadamente con las actividades del proceso que se desarrollan iterativa y concurrentemente, más que de forma secuencial. La parcela del *proceso de ingeniería de requisitos* presenta e introduce las otras cuatro. La subárea de *obtención de requisitos* cubre la captura, descubrimiento o adquisición de éstos. La parte de *análisis* trata de solventar los conflictos entre los requisitos, así como la frontera del sistema. La *validación de requisitos* busca conflictos, omisiones y ambigüedades; a la vez que asegura que los requisitos siguen un estándar de calidad. Por último, la subárea de *gestión de requisitos* permite mantener los requisitos presentes en todo el ciclo de vida, adaptándolos a los cambios sufridos por el proyecto.

Prueba del software: Consiste en la verificación dinámica del comportamiento de los programas ante un conjunto finito de casos de prueba.

Cuadro 4.2. Descripción de las áreas de conocimiento del SWEBOK en la Stone Man Version v0.5

En la Tabla 4.2 se recoge la equivalencia entre las áreas de conocimiento propuestas en las dos versiones de la guía SWEBOK existentes hasta la fecha.

Área de Conocimiento en la Stone Man Version	Áreas de Conocimiento correspondientes en la Straw Man Version	Notas
Gestión de la configuración del software	Gestión de la configuración*	
Construcción del software	Codificación*	
Diseño del software	Diseño detallado*	Se decidió cubrir todo el diseño desde el principio, no distinguiendo entre diseño arquitectónico y diseño detallado. Esta distinción aparece en el estándar 12207
Infraestructura de la Ingeniería del Software	Métodos de desarrollo (orientados al objeto, formales, prototipado) Entornos de desarrollo de software	Los entornos de desarrollo (herramientas) y la reutilización se consideraron los dos elementos principales del proceso de infraestructura. Los métodos se incluyeron porque con frecuencia las herramientas se construyen para implementar métodos determinados
Gestión de la Ingeniería del Software	Proceso de gestión* Medida/Métricas	La definición de este proceso en el estándar 12207 hace mención del uso de datos cuantificables para la toma de decisiones, por eso la parte de *métricas* se ha agrupado con la de *proceso de gestión*
Proceso de Ingeniería del Software	Proceso de mejora*	
Evolución y mantenimiento del software	Proceso de mantenimiento*	
Análisis de la calidad del software	Aseguramiento de la calidad* Verificación y validación* Confiabilidad del software	Todas las áreas relacionadas con la calidad se han agrupado en una sola área de conocimiento
Análisis de los requisitos del software	Análisis de requisitos*	
Prueba del Software	Prueba*	
	Presentación y definición de la Ingeniería del Software	Se incluyó en la Straw Man Version porque aparecía en todos los libros de texto sobre Ingeniería del Software, y se necesita algún tipo de introducción en la Stone Man Version, pero no ha sido incluida en ninguna área como tal. Cuando se establezca el formato final de la Stone Man Version se decidirá como introducirla

* Área de la Straw Man Version basada en el estándar ISO/IEC 12207.

Tabla 4.2. Equivalencia entre la lista de áreas de conocimiento de la Stone Man Version y de la Straw Man Version de la guía SWEBOK [Bourque et al., 1999b]

En la Tabla 4.3 se establece la correspondencia entre las áreas de conocimiento propuestas en la Stone Man Version v0.5 de la guía SWEBOK y el estándar ISO/IEC 12207.

Área de Conocimiento de la Stone Man	Estándar ISO/IEC 12207	
Análisis de los requisitos del software	Análisis de requisitos	PROCESOS PRINCIPALES
Diseño del software	Diseño arquitectónico Diseño detallado	
Construcción del software	Codificación Integración	
Prueba del Software	Prueba Instalación Soporte a la aceptación Proceso de operación Explotación del software Soporte operativo a los usuarios	
Evolución y mantenimiento del software	Proceso de mantenimiento	
Gestión de la configuración del software	Gestión de la configuración	PROCESOS DE SOPORTE
Análisis de la calidad del software	Aseguramiento de la calidad Verificación y validación Revisión conjunta Auditoría	
Gestión de la Ingeniería del Software	Procesos de gestión	PROCESOS DE LA ORGANIZACIÓN
Infraestructura de la Ingeniería del Software	Proceso de infraestructura	
Proceso de Ingeniería del Software	Proceso de mejora	

Tabla 4.3. Correspondencia entre las áreas de conocimiento de la guía SWEBOK Stone Man Version v0.5 y el estándar ISO/IEC 12207 [Bourque et al., 1999b]

SWE-BOK propuesto para la FAA

Este cuerpo de conocimiento (denotado por las siglas SWE-BOK) es el resultado de un trabajo patrocinado por la **FAA** (Federal Aviation Administration) de EEUU como parte de un proyecto para mejorar los procesos de adquisición, desarrollo y mantenimiento del software de dicha entidad.

Este cuerpo de conocimiento pretende contribuir al trabajo que está realizando el SWECC (Software Engineering Coordination Committee) bajo el patrocinio de ACM e IEEE-CS para el desarrollo de la Ingeniería del Software y su madurez como disciplina.

Este cuerpo de conocimiento se recoge en [Hilburn et al., 1999], donde el término *conocimiento* se utiliza para describir el espectro completo del contenido de la

disciplina: *información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos* y *bibliografía*.

Este cuerpo de conocimiento se estructura en tres niveles de abstracción: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*, para lograr así un balance entre la simplicidad y la claridad y el nivel apropiado de detalle en la descripción del conocimiento. En el Cuadro 4.3 se recogen las definiciones que se manejan en esta estructuración, mientras que en la Figura 4.4 se pueden apreciar estos niveles de abstracción de forma gráfica.

Conocimiento: Término utilizado para describir el espectro completo de los contenidos de la disciplina: *información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos* y *bibliografía*.

Cuerpo de conocimiento: Descripción jerárquica del conocimiento sobre la Ingeniería del Software que organiza y estructura el conocimiento en tres niveles de jerarquía: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*.

Categoría de conocimiento: Una subdisciplina de la Ingeniería del Software que es generalmente reconocida como una parte significativa de este cuerpo de conocimiento de la Ingeniería del Software. Son elementos estructurales de alto nivel, utilizados para organizar, clasificar y describir el conocimiento sobre Ingeniería del Software. Cada una de ellas está compuesta por un conjunto de áreas de conocimiento.

Área de conocimiento: Una subdivisión de una categoría de conocimiento que representa el conocimiento de la Ingeniería del Software que está lógicamente cohesionado y relacionado con la categoría de conocimiento mediante la herencia o la agregación. Cada una de ellas está compuesta de un conjunto de unidades de conocimiento.

Unidad de conocimiento: Una subdivisión de un área de conocimiento que representa un componente básico del cuerpo de conocimiento de la Ingeniería del Software que tiene una descripción explícita. Para el propósito de este cuerpo de conocimiento, cada una de estas unidades es atómica; esto es, no se subdivide en elementos más básicos.

Cuadro 4.3. Definiciones manejadas en el SWE-BOK de la FAA [Hilburn et al., 1999]

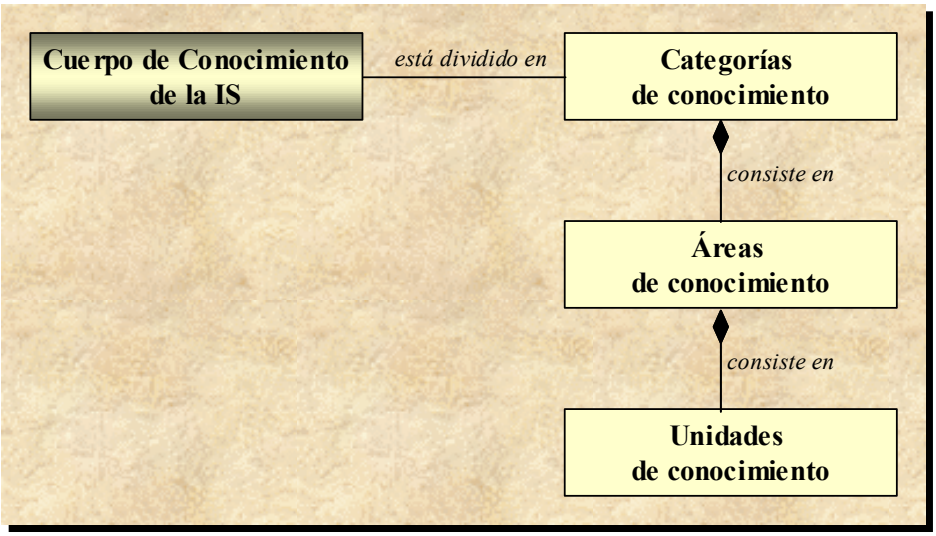

Figura 4.4. Niveles de abstracción en la arquitectura del SWE-BOK de la FAA [Hilburn et al., 1999]

Este cuerpo de conocimiento está formado por cuatro categorías de conocimiento: *Fundamentos de Informática, Ingeniería del Producto Software, Gestión del Software* y *Dominios Software*. En las siguientes tablas se resume este cuerpo de conocimiento.

1. FUNDAMENTOS DE INFORMÁTICA	
Descripción:	*Esta categoría concierne al conocimiento, conceptos, teoría, principios, métodos, propiedades y aplicaciones de Informática que forman la base para el desarrollo de software y de la disciplina de la Ingeniería del Software*
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
1.1 Algoritmos y estructuras de datos	*1.1.1 Estructuras de datos básicas* *1.1.2 Diseño de algoritmos* *1.1.3 Análisis de algoritmos*
1.2 Arquitectura del computador	*1.2.1 Sistemas digitales* *1.2.2 Organización de un sistema computacional* *1.2.3 Arquitecturas alternativas* *1.2.4 Comunicaciones y redes*
1.3 Fundamentos matemáticos	*1.3.1 Lógica matemática y prueba de sistemas* *1.3.2 Estructuras matemáticas discretas* *1.3.3 Sistemas formales* *1.3.4 Combinatoria* *1.3.5 Probabilidad y estadística*
1.4 Sistemas operativos	*1.4.1 Fundamentos de sistemas operativos* *1.4.2 Gestión de procesos* *1.4.3 Gestión de memoria* *1.4.4 Seguridad y protección* *1.4.5 Sistemas distribuidos y de tiempo real*
1.5 Lenguajes de programación	*1.5.1 Teoría de lenguajes de programación* *1.5.2 Paradigmas de programación* *1.5.3 Diseño e implementación de lenguajes de programación*

Tabla 4.4. Categoría de Fundamentos de Informática

2. INGENIERÍA DEL PRODUCTO SOFTWARE	
Descripción:	*Esta categoría se refiere a un conjunto de actividades bien definidas e integradas para producir productos software consistentes. Incluye actividades técnicas, que involucran la documentación de estos productos y el mantenimiento de la traza y la consistencia entre ellos. También se refiere al control de la transición entre las diferentes fases del ciclo de vida del software, así como a las actividades que se necesitan para ofrecer productos software de alta calidad a los clientes.*
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
2.1 Ingeniería de requisitos del software	*2.1.1 Obtención de requisitos* *2.1.2 Análisis de requisitos* *2.1.3 Especificación de requisitos*
2.2 Diseño del software	*2.2.1 Diseño arquitectónico* *2.2.2 Especificación abstracta* *2.2.3 Diseño de la interfaz* *2.2.4 Diseño de las estructuras de datos* *2.2.5 Diseño de algoritmos*
2.3 Codificación del software	*2.3.1 Implementación de código* *2.3.2 Reutilización de código* *2.3.3 Estándares de codificación y documentación*
2.4 Prueba del software	*2.4.1 Pruebas de unidad* *2.4.2 Pruebas de integración* *2.4.3 Pruebas del sistema* *2.4.4 Pruebas de rendimiento* *2.4.5 Pruebas de aceptación* *2.4.6 Pruebas de instalación* *2.4.7 Documentación de la prueba*
2.5 Explotación y mantenimiento	*2.5.1 Instalación y explotación del software* *2.5.2 Operaciones de mantenimiento del software* *2.5.3 Proceso del mantenimiento del software* *2.5.4 Gestión del mantenimiento del software* *2.5.5 Reingeniería del software*

Tabla 4.5. Categoría de Ingeniería del Producto Software

3. GESTIÓN DEL SOFTWARE	
Descripción:	*Esta categoría casa con los conceptos, métodos y técnicas para la gestión de los productos y proyectos software. Incluye actividades relacionadas con la gestión de proyectos, gestión de riesgos, calidad del software y gestión de la configuración.*
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
3.1 Gestión del proyecto software	*3.1.1 Planificación del proyecto* *3.1.2 Organización del proyecto* *3.1.3 Estimación del proyecto* *3.1.4 Calendario del proyecto* *3.1.5 Control del proyecto*
3.2 Gestión de riesgos del software	*3.2.1 Análisis del riesgo* *3.2.2 Planificación de la gestión del riesgo* *3.2.3 Monitorización del riesgo*
3.3 Gestión de la calidad del software	*3.3.1 Aseguramiento de la calidad del software* *3.3.2 Verificación y validación* *3.3.3 Métricas del software* *3.3.4 Sistemas dependientes*
3.4 Gestión de la configuración software	*3.4.1 Identificación de la configuración del software* *3.4.2 Control de la configuración del software* *3.4.3 Auditoría de la configuración del software* *3.4.4 Contabilidad del estado de la configuración del software*
3.5 Gestión del proceso software	*3.5.1 Gestión cuantitativa del proceso del software* *3.5.2 Mejora del proceso del software* *3.5.3 Evaluación del proceso software* *3.5.4 Automatización del proceso software* *3.5.5 Ingeniería del proceso software*
3.6 Adquisición del software	*3.6.1 Gestión de la obtención* *3.6.2 Planificación de la adquisición* *3.6.3 Gestión del rendimiento*

Tabla 4.6. Categoría de Gestión del Software

4. DOMINIOS SOFTWARE	
Descripción:	*Esta categoría tiene que ver con el conocimiento de dominios específicos que involucran la utilización y aplicación de la Ingeniería del Software*
ÁREAS DE CONOCIMIENTO	
4.1 Inteligencia artificial	
4.2 Sistemas de bases de datos	
4.3 Interacción hombre-máquina	
4.4 Computación numérica y simbólica	
4.5 Simulación por computadora	
4.6 Sistemas de tiempo real	

Tabla 4.7. Categoría de Dominios Software

SE-BOK propuesto por el WGSEET

El fin último del **WGSEET** (*Working Group on Software Engineering Education and Training*) es desarrollar un modelo de currículo para la Ingeniería del Software que pueda ser aplicado en todo, o en parte, en el desarrollo de programas educativos especializados en Ingeniería del Software. Esta propuesta se recoge en [Bagert et al., 1999] (y de forma más esquemática en [Hilburn et al., 1998]).

Como parte importante de este proyecto está la definición de un cuerpo de conocimiento de la Ingeniería del Software que sirva de base a la propuesta curricular. Este cuerpo de conocimiento (denotado por SE-BOK en este trabajo) se organiza en cuatro áreas de conocimiento: *el área central, el área de fundamentos, el área de conceptos recurrentes* y *el área de soporte*.

SE-BOK [Bagert et al., 1999]	
ÁREAS DE CONOCIMIENTO	**COMPONENTES DE CONOCIMIENTO**
Área Central – *incluye aquellos componentes que definen la esencia de la Ingeniería del Software*	• Requisitos del software • Diseño del software • Construcción del software • Gestión de proyectos software • Evolución del software
Área de Fundamentos – *incluye aquellos componentes que sirve de base a las áreas central y de conceptos recurrentes*	• Fundamentos de Informática • Factores humanos • Dominios de aplicación
Área de Conceptos Recurrentes – *son elementos que discurren por todos los componentes de conocimiento del área central*	• Ética y profesionalismo • Procesos software • Calidad del software • Modelado de software • Métricas del software • Herramientas y entornos • Documentación
Área de Soporte – *incluye otros campos de estudio que ofrecen los conocimientos necesarios para completar la educación de los ingenieros del software*	• Educación general • Matemáticas • Ciencias naturales • Ciencias sociales • Empresariales • Ingeniería • ...

Tabla 4.8. SE-BOK propuesto por el WGSEET

Comparativa

De los diferentes cuerpos de conocimiento que se han presentado, el propuesto por **IEEE-CS/ACM** es el que más respaldo internacional tiene y, al igual que el propuesto por el **WGSEET**, tiene entre sus cometidos servir de base para la definición de una propuesta curricular para la disciplina de la Ingeniería del Software.

El propuesto para la **FAA** es el que abarca unos contenidos más amplios, buscando definir y evaluar las competencias software que se necesitan en una organización con unas dependencias altas de los sistemas software.

El **SWEBOK** de **IEEE-CS/ACM** es la única propuesta que establece la frontera de la Ingeniería del Software identificando otras disciplinas relacionadas, ya que las otras propuestas de una u otra forma introducen en el cuerpo de conocimiento temas relacionados con otras disciplinas. Una de las intersecciones que más se repite es con la parte de **Gestión de Proyectos**, que por otra parte tiene su propio cuerpo de conocimiento [Duncan, 1996].

4.2.4 La enseñanza de la Ingeniería del Software

Tras la revisión anterior sobre los conceptos relacionados con la disciplina de la Ingeniería del Software, se van a efectuar algunas consideraciones de tipo general sobre la enseñanza de la Ingeniería del Software.

Sin duda, una característica de la Ingeniería del Software como disciplina universitaria, frente a otras disciplinas más establecidas, es el dinamismo con el que cambian sus conceptos y herramientas. Por ello es importante atender a las recomendaciones sobre la enseñanza de la Ingeniería del Software que realizan los organismos y organizaciones internacionales relacionados con la Ciencia de la Informática, los Sistemas de Información y la propia Ingeniería del Software. Atender a este tipo de recomendaciones permite que los conocimientos transmitidos y las habilidades adquiridas por los alumnos respondan a las necesidades profesionales reales y no queden obsoletos en poco tiempo.

Del análisis realizado en el apartado anterior se puede deducir que la Ingeniería del Software consiste en un gran número de actividades interrelacionadas que resultan muy difíciles de conjugar bajo un único epígrafe. Por ello, y sobre todo con fines educativos, es imprescindible introducir una cierta organización en esos contenidos, que permita la definición de un programa educativo consistente y ordenado.

En este sentido en [Ardis and Ford, 1989], [Ford, 1991a] se identifican para la formación en Ingeniería del Software los puntos de vista del proceso de ingeniería y de los productos resultantes. A continuación se exponen brevemente las características más importantes de esta doble visión en formación en Ingeniería del Software.

❖ **Punto de vista del proceso**

El proceso de la Ingeniería del Software incluye un amplio rango de actividades realizadas por los ingenieros de software; pero a lo largo de este rango muchos aspectos de estas actividades son similares. Los elementos del proceso pueden considerarse en dos dimensiones: *la actividad* y *el aspecto*.

- *Actividad.* Las actividades del proceso se dividen en cuatro grupos: *desarrollo, control, dirección* y *operaciones*.
    - ***Actividades de Desarrollo.*** Creación o producción del software de los componentes del sistema, incluyendo análisis de requisitos, especificación, diseño, implementación y prueba.
    - ***Actividades de Control***. Estas actividades están más relacionadas con el control del desarrollo que con la producción del software. Las dos actividades principales del control hacen referencia a la evolución del software y a la calidad del software. En este apartado se consideran las actividades relacionadas con la dirección de la configuración, el control de los cambios, el mantenimiento, el control de la calidad, las pruebas, la evaluación, la verificación y la validación.
    - ***Actividades de Dirección***. Implica la dirección ejecutiva, administrativa, y supervisora de un proyecto de software, incluyendo las actividades técnicas que soportan el proceso ejecutivo de decisión. Las actividades que se pueden considerar aquí son las de planificación del proyecto, localización de recursos, organización de equipos de trabajo, estimación de costes y aspectos legales.
    - ***Actividades de Operación***. Relacionado con el uso de los sistemas de software. Estas actividades incluyen formación del personal en el uso del sistema, planificación para la entrega e instalación de sistemas, cambio desde el sistema antiguo (manual o automático) al nuevo, operación del software y retirada del sistema.
- *Aspecto*. Las actividades de la Ingeniería se dividen tradicionalmente en actividades analíticas y sintéticas. Se consideran seis aspectos de estas actividades para recoger esta distinción: *abstracción, representación, métodos, herramientas, medición* y *comunicaciones*.
    - ***Abstracción.*** Incluye los principios fundamentales y los modelos formales (Modelos del proceso de desarrollo del software, máquinas de estados finitos y redes de Petri, modelo COCOMO...).
    - ***Representación***. Incluye notaciones y lenguajes (Ada, Tablas de Decisión, diagramas de flujo, PERT).

- ***Métodos.*** Incluye métodos formales, practicas actuales, y metodologías (OOD, Programación estructurada...).
- ***Herramientas***. Incluye los conjuntos de herramientas software individuales e integradas (e implícitamente los sistemas hardware donde se ejecutan). Pueden mencionarse las de propósito general (correo electrónico y proceso de textos), las herramientas relativas al diseño e implementación (compiladores y editores sensitivos a la sintaxis) y las herramientas de control de proyectos.
- ***Medición***. Los aspectos de medición incluyen análisis de medidas y evaluación de los productos y el proceso software, así como el impacto del software en la organización; en esta categoría hay que incluir las métricas y los estándares. Esta área merece ser tenida en cuenta en la formación ya que los ingenieros de software, al igual que los ingenieros de los campos tradicionales, necesitan conocer qué medir, cómo medirlo y cómo utilizar los resultados para analizar y evaluar cómo progresan los procesos y productos.
- ***Comunicación***. La comunicación es el último aspecto. Todas las actividades de los ingenieros de software implican comunicación tanto oral como escrita, así como producción de documentación. Un ingeniero de software debe tener unas buenas habilidades en las técnicas generales de comunicación y una comprensión de las formas apropiadas de documentación para cada actividad [Levine et al., 1991].

❖ **Punto de vista del producto**

A menudo es conveniente discutir las actividades y aspectos en el contexto de una determinada clase de sistema de software; por ejemplo la programación concurrente y la secuencial tienen características diferenciadoras. Así, se añaden dos nuevas dimensiones a la estructura organizacional del contenido curricular: *las clases de sistemas software* y *los requisitos del sistema*.

- *Clases de sistemas software*. De las distintas clases que pueden ser consideradas, un grupo se define en función de las relaciones del sistema con su entorno, y sus elementos (partes) están descritos por términos tales como procesamiento por lotes, interactivo, reactivo, tiempo real. Otro grupo tiene elementos descritos en términos tales como distribuido, concurrente, o red. Otro esta definido en función de las características internas, tales como orientado a tablas, orientado a procesos o basado en conocimientos. También, se incluyen áreas de aplicación genéricas o específicas, sistemas de aviónica, sistemas de comunicaciones, sistemas operativos, sistemas de base de datos.

- *Requisitos del sistema*. La discusión de los requisitos del sistema generalmente se centra en los requisitos funcionales, pero existen otras categorías que merecen atención. Identificar y reunir esos requisitos es el resultado de las actividades realizadas a lo largo del proceso de Ingeniería del Software. Ejemplos de estos requisitos son: accesibilidad, adaptabilidad, disponibilidad, compatibilidad, exactitud, eficiencia, tolerancia a fallos, integridad, interoperabilidad, mantenibilidad, rendimiento, portabilidad, protección, formalidad, reusabilidad, robustez, seguridad, comprobabilidad y usabilidad.

Otra forma de enfocar la enseñanza de la Ingeniería del Software es mediante un enfoque basado en un proyecto para que el alumno se aproxime al trabajo tal cual ocurre (*o debiera ocurrir*) en la realidad empresarial [Tomayko, 1987], [Shaw and Tomayko, 1991].

En Sicología se distinguen dos tipos de conocimientos: *declarativo* y *procedural* [Norman, 1988]. El primero es fácil de transcribir y de enseñar; sin embargo, el segundo es imposible de transcribir y difícil de enseñar, siendo más sencillo de transmitir mediante demostración y de aprender por la práctica. Muchos de los procesos de la Ingeniería del Software dependen del conocimiento procedural. Por este motivo se recomienda una importante parte de experiencia adquirida mediante proyectos [Ardis and Ford, 1989].

En [Shaw and Tomayko, 1991] se presentan diferentes modelos de cursos de Ingeniería del Software que toman el proyecto como eje conductor de los mismos. Los modelos que se enuncian son:

- ***Modelo de Ingeniería del Software como producto***. Es el modelo al que se ajustan los cursos compuesto exclusivamente por clases teóricas. Son cursos que concentran en, aproximadamente, un cuatrimestre los conceptos de la Ingeniería del Software. Su mayor desventaja es la ausencia de parte práctica, y por tanto de experiencia. Estos cursos se adecuan a lo que se denomina *énfasis por el ciclo de vida*, que se ajusta a la forma de organizar los libros de texto sobre Ingeniería del Software, especialmente siguiendo el ciclo de vida en cascada, como claramente se aprecia en el libro de **Richard Fairley** [Fairley, 1985] o en ediciones anteriores de libros tan clásicos como el de **Roger S. Pressman** [Pressman, 1987] o **Ian Sommerville** [Sommerville, 1989].
- ***Modelo de la aproximación por seminarios.*** Es similar al anterior en el sentido de que la base del curso son las clases teóricas, pero se distingue en que en este modelo de curso, se reserva un tiempo para que los alumnos presenten trabajos que ellos mismos han realizado sobre algún tema concreto, manejando la bibliografía oportuna.

- ***Modelo de proyecto para grupos pequeños.*** Este modelo de curso incluye la realización de un proyecto de pequeñas dimensiones como parte del curso. Es muy seguido porque divide el curso en trabajo de clase y trabajo de proyecto. Los alumnos se dividen en grupos de entre tres y cinco personas, debiendo abordar un proyecto que les sea familiar y puedan terminar en el tiempo asignado al curso. Este curso permite obtener algunas experiencias derivadas de la aplicación de lo explicado en las clases teóricas, pero es deficiente en el sentido de que no les entrena para el trabajo en proyectos grandes.
- ***Modelo de proyecto para grandes equipos.*** Es la mejor opción para aprender las técnicas que se utilizan en los proyectos reales. La idea es realizar un proyecto con toda la clase. Típicamente se elige un proyecto consistente en el desarrollo de un producto software, idealmente destinado a un cliente real. Los alumnos se organizan en un solo equipo de desarrollo, asumiendo cada uno de ellos el rol que le sea asignado, y que coincidirá con los roles que aparecen en los entornos industriales reales. Cada alumno mantendrá el rol durante todo el curso, y aprenderá las actividades de los otros roles a través de la interacción con los otros miembros del equipo. Es inviable de llevar a cabo cuando el número de alumnos es alto y la asistencia no es obligatoria. Además, la relación con las actividades propias de otros roles no es tan intensa como las que uno lleva a cabo.
- ***Modelo de proyecto único.*** El curso entero se dedica a la realización de un proyecto. Suele llevarse a cabo cuando la Ingeniería del Software se divide en dos asignaturas independientes, una dedicada por completo a la teoría y la otra a la práctica.

Con independencia del modelo que se siga existen varios tipos de proyectos. En [Shaw and Tomayko, 1991] se hace un repaso de ellos, encontrando:

- ***El proyecto de "juguete".*** La clase se divide en equipos de 3 a 5 personas; cada equipo recibe una labor predefinida por el responsable de la asignatura. Puede existir una variante donde cada equipo crea su propia especificación. Las ventajas son que los alumnos aplican las enseñanzas de la Ingeniería del Software trabajando en equipo, aunque no se satisfagan los requisitos por completo o no se llegue a terminar la implementación, y los proyectos sean fáciles de gestionar. Se puede llegar a situaciones en las que alumnos de segundo ciclo actúen como gestores de los proyectos realizados por alumnos de primer ciclo. La mayor desventaja es que se omiten técnicas propias de los proyectos grandes como puede ser la gestión de la configuración. Es una práctica muy extendida.
- ***Proyecto basado en componentes.*** Los alumnos se organizan en grupos que desarrollan componentes software. Se establecen después una serie de

proyectos que se llevarán a cabo con los componentes realizados, para ello debe haber una primera etapa de *adquisición de componentes*, una segunda de integración de los mismos y una tercera de *modificación* o *adaptación* a las necesidades del grupo. Es un tipo de proyecto en el que se manejan técnicas propias de proyectos grandes, y los alumnos se acostumbran a diseñar componentes reutilizables. Su mayor desventaja está en su gestión, muy parecida al caso anterior.

- ***Proyecto para un cliente externo.*** Los estudiantes trabajan en componentes que deberán integrarse para la realización de un producto para un cliente externo, que deberá pasar los criterios de aceptación oportunos acordados con el cliente. Se puede llevar a cabo en pequeños equipos o con un gran equipo. Es el que más se acerca a la realidad, debiendo hacer uso de todas las técnicas propias de un proyecto de grandes dimensiones, donde la relación con el cliente será uno de los puntos más sobresalientes.
- ***Proyectos individuales.*** Cada equipo tiene un proyecto diferente. Los equipos pueden tener clientes externos, que con frecuencia son facilitados por el responsable de la asignatura. Puede convertirse en una caos desde la perspectiva de gestión de la globalidad de los proyectos.

4.3 Programación Orientada a Objetos

4.3.1 Introducción

Al igual que se hizo con la Ingeniería del Software, se va a realizar un repaso global por la Orientación a Objetos, presentando su marco histórico y conceptual.

La Orientación a Objetos forma parte de la Ingeniería del Software como un paradigma de desarrollo con su propio marco conceptual que involucra terminología, métodos, modelos, procedimientos, técnicas, prácticas y procesos.

Los métodos de la Orientación a Objetos han sido reconocidos en el ámbito de las tecnologías de la información, como la mejor filosofía para abordar la reutilización y la extensibilidad del software.

En una primera aproximación se puede decir que el término *orientado a objetos* significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como un comportamiento. Esto se opone frontalmente a la programación convencional, donde las estructuras de datos y la funcionalidad sólo se relacionan débilmente.

Tradicionalmente se asocia la Orientación a Objetos con la Programación Orientada a Objetos, es decir centrando la atención en los lenguajes de programación. Cierto es que los lenguajes de programación orientados a objetos son útiles para eliminar algunas

restricciones propias de los lenguajes de programación tradicionales. Sin embargo, enfatizar la fase de codificación supone un retroceso de la Ingeniería del Software al priorizar los mecanismos de implementación frente al proceso de pensamiento subyacente al cual sirven de base [Rumbaugh et al., 1991]. Así, este paradigma se extiende a todas las fases del ciclo de vida con un modelo común subyacente a todas estas fases: *el modelo objeto*.

Se entenderá, entonces, por *desarrollo orientado a objetos* la forma de pensar acerca del software basándose en abstracciones del mundo real; donde la palabra desarrollo hace alusión directa al bloque inicial de fases del ciclo de vida del software: *análisis, diseño* e *implementación.*

Aparecen así los conceptos de **Programación Orientada a Objetos** (POO)**, Diseño Orientado a Objetos** (DOO) y **Análisis Orientado a Objetos** (AOO)**.**

> **"La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia"**
>
> **Grady Booch, [Booch, 1994]**

Cabe destacar tres partes importantes en esta definición:

1. La POO utiliza *objetos*, no algoritmos, como sus bloques lógicos de construcción fundamentales.
2. Cada objeto es una *instancia* de alguna *clase*.
3. Las clases están relacionadas con otras clases por medio de relaciones de *herencia*.

Mientras que los métodos de programación ponen el énfasis en el uso correcto y efectivo de mecanismos particulares del lenguaje de programación que se utiliza, los métodos de diseño enfatizan la estructuración correcta y efectiva de un sistema complejo, definiéndose como sigue.

> **"El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para descubrir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña"**
>
> **Grady Booch, [Booch, 1994]**

Se destacan dos aspectos de esta definición:

1. El DOO da lugar a una descomposición orientada a objetos.
2. Utiliza diversas notaciones para expresar diferentes modelos del diseño lógico (*estructura de clases y objetos*) y físico (*arquitectura de módulos y*

procesos) de un sistema, además de aspectos estáticos y dinámicos del sistema.

El modelo de objetos ha influido en las fases iniciales del ciclo de vida del desarrollo del software. El análisis orientado a objetos enfatiza la construcción de modelos del mundo real, utilizando una visión del mundo orientada a objetos, pudiéndose definir como:

> **"El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de las clases y los objetos que se encuentran en el vocabulario del dominio del problema"**
>
> **Grady Booch, [Booch, 1994]**

Toda la Orientación a Objetos gira en torno a dos conceptos fundamentales: *el objeto* y *la clase*. Ambos son dos conceptos estrechamente relacionados, pero que no deben confundirse nunca [Holland et al., 1997]. Para terminar esta introducción se van a presentar algunas de las definiciones que, tanto de objeto como de clase, pueden encontrarse en la bibliografía especializada.

Un objeto modela una parte de la realidad. Con el concepto de objeto, se modela la permanencia e identidad de conceptos percibidos. Así un objeto puede definirse como:

> **"Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema"**
>
> **[Smith and Tockey, 1988]**

> **"Una colección de operaciones que comparten un estado"**
>
> **Peter Wegner, [Wegner, 1990]**

> **"Concepto, abstracción, o cosa con frontera y significado débil, perteneciente al problema que se trata; instancia de una clase"**
>
> **[Rumbaugh et al., 1991]**

> **"Un objeto es un encapsulado de un conjunto de operaciones o métodos que pueden ser invocados externamente y de un estado que puede recordar los efectos de los métodos"**
>
> **[Blair et al., 1991]**

> **"Entidad conceptual que es identificable, tiene características que comporten un estado interno y tiene unas operaciones que pueden cambiar el estado del sistema local, y que también pueden solicitar operaciones de objetos relacionados"**
>
> **[Champeaux et al., 1993]**

> **"Un objeto tiene estado, comportamiento e identidad; la estructura y el comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables"**
>
> **Grady Booch, [Booch, 1994]**

> **"Un concepto, abstracción o cosa que puede ser individualmente identificada y tiene significado en una aplicación. Un objeto es una instancia de una clase"**
>
> **[Blaha and Premerlani, 1998]**

> **"Una entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento es representado por sus operaciones, métodos y máquinas de estados. Un objeto es una instancia de una clase"**
>
> **[OMG, 1999]**

Por su parte las clases sirven como plantillas para la creación de objetos, especificando un comportamiento a todas sus instancias. Se puede definir como:

> **"Una clase es una plantilla desde la que sus objetos pueden ser creados. Contiene la definición de los descriptores de estado y los métodos de los objetos"**
>
> **[Blair et al., 1991]**

> **"Es un conjunto de objetos que comparten una estructura común y un comportamiento común"**
>
> **Grady Booch, [Booch, 1994]**

> **"Descripción abstracta de los datos y del comportamiento de una colección de objetos similares"**
>
> **Timothy Budd, [Budd, 1991]**

> **"Descripción de un grupo de objetos con propiedades similares, comportamientos comunes, interrelaciones comunes y semántica común"**
>
> **[Rumbaugh et al., 1991]**

> **"Una clase es un tipo abstracto de datos equipado con una posible implementación"**
>
> **Bertrand Meyer, [Meyer, 1997]**

La Orientación a Objeto se distribuye en las dos asignaturas objeto de este proyecto docente de la siguiente forma: el AOO entrará a formar parte del temario de la asignatura de ***Ingeniería del Software***, mientras que la parte de POO y especialmente el DOO serán el cometido de la asignatura de la ***Programación Orientada a Objetos***.

4.3.2 Marco histórico de la Orientación a Objetos

Los pasos por los que ha pasado la evolución histórica de la Orientación a Objeto han sido similares a los de otros métodos de desarrollo; esto es, en primer lugar el énfasis estuvo centrado en las técnicas de programación, para posteriormente ir centrando la atención en las fases de desarrollo de mayor nivel de abstracción, diseño y análisis, y actualmente en los procesos de negocio [Pancake, 1995].

Lo que no ha sido tan normal ha sido su evolución temporal, pues tras los primeros pasos dados a finales de la década de los sesenta, sufrió una parada significativa en su evolución hasta la segunda mitad de la década de los ochenta (como dato significativo, entre junio de 1978 y diciembre de 1985, sólo aparecen nueve artículos – de unos cuatrocientos – en ACM SIGPLAN Notices, mencionando la tecnología orientada a objetos de una forma significativa [Sharp et al., 2000]), que resurge con tremenda fuerza, especialmente a partir de la primera conferencia OOPSLA en 1986, dando lugar a una gran diversidad, que podría calificarse de caótica en algunas parcelas (como por ejemplo los métodos de análisis y diseño). En los últimos años de la década de los noventa se ha producido una cierta madurez en la tecnología de objetos, que camina hacia la unificación y los primeros estándares.

El repaso que se va a hacer por la historia de la Orientación a Objetos se ha divido en tres apartados. En el primero de ellos se va a buscar el origen del concepto de objeto. En el segundo se va a sintetizar la historia de los lenguajes de programación orientados a objetos. Por último, en el tercero se va a presentar la evolución de los métodos de desarrollo en la tecnología de objetos.

4.3.2.1 Origen del concepto de objeto

El término *objeto* surge de forma independiente en varios campos de la informática, casi simultáneamente a principios de los setenta, para referirse a nociones que eran diferentes en su apariencia pero relacionadas entre sí. Todas estas nociones se inventaron para manejar la complejidad de los sistemas software de tal forma que los objetos representaban componentes de un sistema descompuesto modularmente o bien unidades modulares de representación del conocimiento [Yonezawa and Tokoro, 1987].

Levy añade que los siguientes acontecimientos contribuyeron a la evolución de los conceptos orientados a objetos [Levy, 1984]:

- *Avances en la arquitectura de los computadores, incluyendo los sistemas de capacidades y el apoyo en hardware para conceptos de sistemas operativos.*
- *Avances en los lenguajes de programación, como se demostró en Simula, Smalltalk, CLU y Ada.*
- *Avances en metodología de la programación, incluyendo la modularización y la ocultación de la información.*

A esta lista de contribuciones podrían añadirse las tres siguientes [Booch, 1994]:

- *Avances en los modelos de bases datos.*

- *Investigación en Inteligencia Artificial.*
- *Avances en Filosofía y Ciencia Cognitiva.*

El concepto de un objeto tuvo sus inicios en el hardware con la aparición de arquitecturas basadas en descriptores y, posteriormente, arquitecturas basadas en capacidades [Ramamoorthy and Sheu, 1988]. Estas arquitecturas representaron una ruptura con las arquitecturas clásicas de **Von Neumann**, surgiendo como consecuencia de los intentos realizados para eliminar el hueco existente entre las abstracciones de alto nivel del los lenguajes de programación y las abstracciones de bajo nivel de la propia máquina. Según sus inventores estas arquitecturas ofrecen ventajas del tipo: *mejor detección de errores, mejora de la eficiencia de ejecución, menos tipos de instrucciones, compilación más sencilla* y *reducción de los requisitos de almacenamiento*. Algunos computadores con arquitectura orientada a objetos fueron el Burroughs 5000, el Plessey 250, el Intel 432, el IBM System/38 o el Rational R10000.

Muy relacionados con las arquitecturas orientadas a objetos están los sistemas operativos orientados a objetos. El trabajo de **Dijkstra** con el sistema de multiprogramación THE fue el primero que introdujo la idea de construir sistemas como máquinas de estados en capas [Dijkstra, 1968]. Otros sistemas operativos pioneros en la tecnología de objetos fueron UCLA Secure UNÍS (para el PDP 11/45 y 11/70), StarOS y Medusa (para Cm* de CMU) o el iMAX (para el Intel 432). Sistemas operativos actuales, como Microsoft Windows NT, parecen seguir el camino de los objetos.

Sin duda alguna la contribución más importante al modelo de objetos estriba en los denominados lenguajes de programación basados en objetos y orientados a objetos. Las ideas fundamentales de clase y objeto aparecieron por primera vez en el lenguaje Simula 67.

En 1972 **David Parnas** introduce la idea de ocultación de la información [Parnas, 1972], y en la década de los setenta varios investigadores, destacando **Liskov y Zilles** [Liskov and Zilles, 1977], **Guttag** [Guttag, 1980] y **Mary Shaw** [Shaw, 1984], fueron pioneros en el desarrollo de mecanismos de tipos de datos abstractos. **Hoare** contribuyó a esos desarrollos con su propuesta de una teoría de tipos y subtipos.

Aunque la tecnología de Bases de Datos ha evolucionado un tanto independientemente de la Ingeniería del Software, también ha contribuido al modelo de objetos [Atkinson and Buneman, 1987], especialmente mediante las ideas de la aproximación entidad-relación al modelado de datos [Rumbaugh, 1988]. En el modelo entidad-relación, propuesto inicialmente por **Peter Chen** [Chen, 1976], el mundo se modela en términos de sus entidades, los atributos de éstas y las relaciones entre esas entidades.

En el campo de la Inteligencia Artificial, los avances en representación del conocimiento han contribuido a una comprensión de las abstracciones orientadas a objetos. En 1975, **M. Minsky** propuso por primera vez una teoría de marcos para

representar objetos del mundo real tal y como los perciben los sistemas de reconocimiento de imágenes y el lenguaje natural [Barr and Feigenbaum, 1981]. Desde entonces se han utilizado los marcos como fundamento arquitectónico para diversos sistemas inteligentes.

Por último la Filosofía y la Ciencia Cognitiva han contribuido al avance del modelo de objetos. La idea de que el mundo podía verse en términos de objetos o procesos procede de los filósofos griegos, más concretamente de la Teoría de las Ideas desarrollada por **Platón**, a partir de las enseñanzas de **Sócrates**, y estudiada y ampliada por **Aristóteles** (en la tabla se tiene una equivalencia entre los conceptos de la Teoría de las Ideas y la Orientación a Objetos, para una mayor información consultar [Alhir, 1998]). Así se puede decir que la Orientación a Objetos se acerca más al enfoque Aristotélico-Tomista[20], frente al enfoque Kantiano de los métodos estructurados. Asimismo, en el siglo XVII se tiene a **Descartes** defendiendo que los seres humanos aplican de forma natural una visión orientada a objetos del mundo [Stillings et al., 1987]. Ya en el siglo XX, **Rand** amplía estos conceptos en su filosofía de la epistemología objetivista [Rand, 1979].

TEORÍA DE LAS IDEAS	PARADIGMA OBJETUAL
Método Socrático (la recolección)	Abstracción (*como forma de obtener conocimiento*)
Ideas, universalidad y sustancias secundarias	Clases (*y encapsulación*)
Cosas, particulares y sustancias primarias	Objetos (*y encapsulación)*
Procedimiento de captura y división y el principio de uno sobre muchos	Herencia (*y polimorfismo*)
El argumento del tercer hombre	Abstracción (*como marco conceptual para el modelado que involucra múltiples niveles de abstracción*)

Tabla 4.9. Correspondencia entre la Teoría de las Ideas y la Orientación a Objeto

4.3.2.2 Historia de los lenguajes de programación orientados a objetos

Los lenguajes de programación de alto nivel pueden agruparse en cuatro generaciones [Booch, 1994]; según soporten abstracciones matemáticas, algorítmicas, de datos u orientados a objetos. Un lenguaje se considera *basado en objetos* si soporta directamente abstracción de datos y clases, mientras que un lenguaje *orientado a objetos* es aquél que, además de estar basado en objetos, proporciona soporte para la herencia y el polimorfismo.

[20] En la realidad no existen datos o procesos independientes. Santo Tomás de Aquino sostiene que el número, considerado abstractamente, no existe fuera de la mente humana; "*la verdad consiste en la adecuación del entendimiento con las cosas*" (Suma Teológica. I, c.16, a.3.).

Aristóteles indica que "*no se pueden separar, a los objetos en movimiento, por ejemplo*" puesto que "*hay una multitud de accidentes que son esenciales a las cosas, en tanto que cada uno de ellos reside esencialmente en ellas*" (Metafísica. 7ª edición, Madrid, Espasa-Calpe, 1972, p.284.).

El primer lenguaje orientado a objetos data de finales de la década de los sesenta fue **Simula 67** [Dahl et al., 1970], [Dahl and Hoare, 1972], [Birtwistle et al., 1973]. Este lenguaje fue diseñado en 1967 por **Ole-Johan Dhal** y **Kristen Nygaard** en el *Norwegian Computing Center* en Oslo. Ellos partieron de un lenguaje denominado Simula 1 [Dahl and Nygaard, 1966], que fue desarrollado en 1962 para la realización de simulaciones discretas; siendo un conjunto de procedimientos más un preprocesador para el lenguaje **ALGOL 60**. El lenguaje Simula al que se le atribuye el *honor* de ser el primer lenguaje orientado a objetos es el Simula 67. El nombre que se le dio intentaba no romper la continuidad con el primer lenguaje, manteniendo así el enlace con la comunidad de usuarios existentes, pero, como el propio **Nygaard** reconoce, fue un error porque aunque Simula 67 era un lenguaje de carácter general, todo el mundo lo asociaba a un lenguaje para llevar a cabo simulaciones de eventos discretos solamente. El nombre de Simula 67 fue acortado a Simula en 1986, existiendo un estándar del lenguaje desde 1987 [SIS, 1987]. La historia de Simula contada por sus creadores se encuentra en [Nygaard and Dahl, 1981].

Uno de los descendientes directos de Simula fue **Beta** [Madsen et al., 1993], diseñado en Escandinavia con la colaboración de **Kristen Nygaard**. Introduce una construcción denominada *patrón* para unificar los conceptos de clase, procedimiento, función, tipo y corrutina.

Uno de los lenguajes de programación que adoptó los conceptos de clase y mensaje propios de Simula fue **Smalltalk**. Se desarrolla a principios de la década de los setenta en el *Xerox PARC* (*Palo Alto Research Center*) como fruto de una labor sinérgica de un grupo de trabajo, en el que aparecen nombres tan importantes como **Alan Kay, Adele Goldberg** o **Daniel Ingalls** como principales responsables. Smalltalk representa tanto un lenguaje de programación como un entorno de desarrollo de software. Se le considera un lenguaje orientado a objetos puro, en el sentido de que todo en él se ve como un objeto (*desde las propias clases hasta los tipos de datos básicos de otros lenguajes, como puede ser el tipo entero, son objetos*).

Los conceptos de Smalltalk no sólo han influido decisivamente en el diseño de posteriores lenguajes de programación orientados a objetos, sino también en el aspecto y sensación de interfaces gráficas de usuario como las de **Macintosh**, **Windows** o **Motif**, o en la definición de una de las arquitecturas básicas de los entornos gráficos de usuario, la ***arquitectura Modelo-Vista-Controlador (MVC)*** [Krasner and Pope, 1988]**.**

Como lenguaje combina la influencia de Simula con un estilo libre, sin ataduras de tipos, propio de **Lisp**. Hace un gran énfasis en la ligadura dinámica, no haciendo ningún chequeo de tipos.

Hay cinco versiones identificables de Smalltalk, referenciadas por su año de aparición: Smalltalk-72 [Goldberg and Kay, 1976], Smalltalk-74, Smalltalk-76 [Ingalls, 1978], Smalltalk-78 y Smalltalk-80 [Goldberg and Robson, 1983]. Las dos primeras sentaron gran parte de los fundamentos del lenguaje, incluyendo las ideas de paso de

mensajes y polimorfismo. Las versiones posteriores convirtieron a las clases en *ciudadanos de primera clase*, completando la visión de que todo lo que hay en el entorno puede tratarse como un objeto. Hay un importante dialecto de Smalltalk proporcionado por Digital, Smalltalk/V, muy parecido a Smalltalk y disponible para máquinas IBM PC (entornos Windows y OS/2) y Macintosh.

Las referencias por excelencia de Smalltalk son [Goldberg and Robson, 1983], [Goldberg, 1985], [Lalonde and Pugh, 1990] y [Lalonde and Pugh, 1990].

No todos los lenguajes orientados a objetos fueron creados desde cero, sino que muchos han visto la luz como evolución de otro ya existente. Quizás los ejemplos más destacados sean **Object Pascal, Objective-C, C++** y **Ada 95**.

Object Pascal fue diseñado por desarrolladores de **Apple Computer** (algunos de los cuales estuvieron implicados en el desarrollo de Smalltlak), en conjunción con **Niklaus Wirth**, el padre de Pascal. El antecesor inmediato de Object Pascal fue *Clascal*, una versión orientada a objetos del Pascal para el computador *Lisa*. Object Pascal se puso a disposición del público en 1986 y fue el primer lenguaje de programación orientado a objetos soportado por el Macintosh Programmer's Workshop (MPW), el entorno de desarrollo para la familia Apple de computadores Macintosh.

Object Pascal es el esqueleto de un lenguaje orientado a objetos. No proporciona métodos de clase, variables de clase, herencia múltiple ni metaclases. Conceptos que se excluyen de forma deliberada en un intento de suavizar la curva de aprendizaje de los que se aproximan por primera vez a la programación orientada a objeto [Schmucker, 1986].

La referencia principal para Object Pascal es [Apple, 1989]. En la actualidad, la orientación a objetos en derivados del Pascal sigue viva gracias a los esfuerzos de Borland (ahora Inprise) por incorporar las extensiones de objetos a su Turbo Pascal (versiones para MS-DOS y MS - Windows) y al desarrollar posteriormente Delphi (para entornos Windows), cuya versión actual es la 5.0.

Objective-C [Cox, 1984], [Cox and Novobilski, 1990] fue diseñado por **Brad J. Cox** en Stepstone Corporation. Es una definición ortogonal a Smalltalk sobre la base conceptual del lenguaje C. Fue el lenguaje de programación fundamental para el sistema operativo y las estaciones de trabajo NEXTSTEP. Aunque en parte relegado a un segundo plano por el éxito de C++, este lenguaje aún retiene un importante número de usuarios activos.

Como sucede en Smalltalk, Objective-C, pone un énfasis especial en el polimorfismo y en la ligadura dinámica, aunque las versiones actuales han dado un salto atrás con respecto al modelo original de Smalltalk para ofrecer tipos estáticos como una opción (y para algunos de ellos también ligadura estática).

C++ fue desarrollado por **Bjarne Stroustrup** en los AT&T Bell Laboratories a principios de la década de los ochenta. En el Cuadro 4.4 se recogen las fechas más significativas en la evolución de C++.

Figura 4.5. Bjarne Stroustrup

Es un lenguaje híbrido, con comprobación estricta de tipos, en el cual algunas entidades son objetos y otras no. Es una extensión de C, que además de añadir capacidades orientadas a objetos, sirve para mejorar algunas deficiencias del lenguaje C.

1979	Mayo	Comienza el trabajo con C con Clases
	Octubre	La primera implementación de C con Clases en uso
1980	Abril	Primer artículo interno para Bell Labs sobre C con Clases
1982	Enero	Primer artículo externo sobre C con Clases
1983	Agosto	Primera implementación de C++ en uso
	Diciembre	Se le da el nombre de C++
1984	Enero	Primer manual de C++
1985	Febrero	Primera versión externa de C++ (Release E)
	Octubre	Cfront Release 1.0 (primera versión comercial)
	Octubre	The C++ Programming Language [Stroustrup, 1986]
1986	Agosto	The "*whatis paper*" [Stroustrup, 1986b]
	Septiembre	Primera conferencia OOPSLA
	Noviembre	Primer *port* comercial a PC de Cfront (Cfront 1.1, Glockenspiel)
1987	Febrero	Cfront Release 1.2
	Noviembre	Primera conferencia USENIX (Santa Fe, Nuevo Mexico)
	Diciembre	Primera versión de GNU C++ (1.13)
1988	Enero	Primera versión de Oregon Software C++
	Junio	Primera versión de Zortech C++
	Octubre	Primer *workshop* USENIX de desarrolladores en C++
1989	Junio	Cfront release 2.0
	Diciembre	Reunión organizativa ANSI X3J16 (Washington, DC)
1990	Mayo	Primera versión de Borland C++
	Mayo	Primera reunión técnica ANSI X3J16 (Somerset, NJ)
	Mayo	The Annotated C++ Reference Manual [Ellis and Stroustrup, 1990]
	Julio	Plantillas aceptadas (Seattle, WA)
	Noviembre	Excepciones aceptadas (Palo Alto, CA)
1991	Junio	The C++ Programming Language 2nd [Stroustrup, 1991]
	Junio	Primera reunión ISO WG21 (Lund, Sweden)
	Octubre	Cfront release 3.0 (incluye plantillas)
1992	Febrero	Primera versión del DEC C++ (incluye plantillas y excepciones)
	Marzo	Primera versión de Microsoft C++
	Mayo	Primera versión de IBM C++ (incluye plantillas y excepciones)
1993	Marzo	RTTI aceptado (Munich, Alemania)
	Julio	Espacios de nombres aceptado (Munich, Alemania)
1994	Agosto	Registro del *draft* del comité ANSI/ISO
1997	Noviembre	Aprobado el estándar internacional del lenguaje de programación C++

Cuadro 4.4. Fechas relevantes en la evolución de C++

El antecesor inmediato de C++ fue el denominado *C con Clases*, desarrollado por el propio Stroustrup en 1980, recibiendo enormes influencias de los lenguajes Simula y C.

Han existido varias versiones principales del lenguaje C++. La versión 1.0 (y sus versiones menores) añadían características básicas de orientación a objetos al C, como la herencia simple y el polimorfismo, además de la comprobación de tipos y sobrecarga. La versión 2.0, que aparece en 1989, mejoró las versiones anteriores de diversas formas, como la introducción de la herencia múltiple, sobre la base de una amplia experiencia con el lenguaje por parte de una comunidad de usuarios relativamente grande. La versión 3.0, aparecida en 1991, introduce las plantillas (*manejo de clases parametrizadas*) y el manejo de excepciones. Las últimas aportaciones al estándar de C++ han venido de la mano de la incorporación de espacios de nombres e identificación de tipos en tiempo de ejecución. En noviembre de 1997 se aprobó el estándar internacional de C++, que queda recogido en [ISO/IEC, 1998], teniéndose acceso libre a algunos de los borradores de los informes técnicos que se han ido elaborando, por ejemplo el de diciembre de 1996 en [X3J16/WG21, 1996].

En [Stroustrup, 1994] se recogen de mano del padre de C++ las decisiones de diseño tomadas para el desarrollo del lenguaje.

La tecnología inicial de traductores para C++ implicaba el uso de un preprocesador para C, llamado *cfront*. Puesto que este traductor generaba código C como representación intermedia, era posible transportar C++ a prácticamente todas las arquitecturas UNIX con bastante sencillez. Ahora, están disponibles los traductores de C++ y compiladores nativos para los conjuntos de instrucciones de casi todas las arquitecturas.

La bibliografía de referencia sobre C++ es muy abundante; de los numerosos títulos existentes se van a destacar dos [Ellis and Stroustrup, 1990] y [Stroustrup, 1997][21]. Como punto discordante se recomienda la lectura de [Joyner, 1996], donde se hace un recorrido crítico de C++ como lenguaje de programación orientado a objetos, comparándolo con otros lenguajes orientados a objetos.

Eiffel [Meyer, 1992] es un lenguaje orientado a objetos diseñado por **Bertrand Meyer**. Fue ideado no sólo como un lenguaje de programación, sino también como una herramienta de Ingeniería del Software. Aunque se diseñó desde cero, recibe influencias de Simula.

El lenguaje soporta ligadura dinámica y comprobación estática de tipos, ofreciendo flexibilidad en el diseño de la interfaz de una clase, pero aprovechando la seguridad respecto al tipo que proporciona la comprobación estática de tipos. Hay varias características significativas que apoyan el desarrollo de software seguro y de calidad, incluyendo clases parametrizadas, aserciones y excepciones.

[21] Esta edición se ajusta al estándar aprobado.

Como otros lenguajes anteriores a la Orientación a Objetos, Lisp ha servido como influencia para muchos lenguajes de programación orientados a objetos, lo cual no es de extrañar porque Lisp ofrece muchos mecanismos que ayudan a la implementación de los conceptos de la Orientación a Objetos: *una aproximación dinámica para la creación de objetos, gestión automática de memoria con "recolección de basura", fácil implementación de estructuras arbóreas, selección de operaciones en tiempo de ejecución...*

En la década de los ochenta hubo tres frentes que coparon la atención de la orientación a objeto en torno a Lisp: **Loops** [Bobrow and Stefik, 1982], desarrollado por **Xerox**, inicialmente para el entorno *Interlisp*; **Flavors** [Cannon, 1980], desarrollado en el **MIT**, disponible en varias arquitecturas orientadas a Lisp; **Ceyx** [Hullot, 1984], desarrollado en el **INRIA**.

Loops introdujo un concepto interesante: *la programación orientada a los datos*; de forma que se podía asociar una rutina a cada elemento de datos (por ejemplo a un atributo). La ejecución de la rutina era disparada no sólo por la llamada explícita, sino también cuando el elemento era accedido o modificado. Esto abría la puerta a la computación conducida por eventos y a arquitecturas software más descentralizadas.

La unificación de estas propuestas vino de la mano de **CLOS** (*Common Lisp Object Oriented System*) [Paepcke, 1993], una extensión orientada a objetos de Common Lisp, convirtiéndose en el primer lenguaje orientado a objetos en contar con un estándar ANSI.

Aunque fue implementado de forma híbrida en un principio, CLOS está tan bien integrado con las características de Common Lisp que tiene la mayoría de las ventajas de los lenguajes de programación orientados a objetos puros. Esto se debe a que todos los objetos de datos, incluyendo los átomos y las listas de Lisp, son miembros de una clase. Los métodos correspondientes a primitivas de Lisp pertenecen a una estructura de herencia. Como resultado, no hay una distinción práctica entre primitivas de Lisp y objetos.

CLOS ofrece una rica colección de metadatos a los cuales se puede acceder y pueden ser actualizados en tiempo de ejecución. Se pueden definir clases nuevas, y se pueden añadir dinámicamente métodos a las clases.

El sistema del lenguaje CLOS no impone el concepto de encapsulación. Se recomienda a los programadores que definan y documenten la interfaz pública de todas las clases, y que utilicen únicamente las características indicadas de las otras clases, pero nada impide que el código de una clase acceda directamente a los detalles de implementación de otra clase. Esta falta de aplicación de las convenciones concuerda con la política general de Lisp, consistente en ofrecer la mayor flexibilidad y el mayor espacio posible para la experimentación.

La última revolución dentro del seno de los lenguajes de programación orientados a objetos ha sido **Java**. Desarrollado por un equipo de **Sun Microsystems**, este lenguaje ha provocado ríos de tinta desde su aparición en escena a finales de 1995, especialmente por su íntima relación con Internet.

Los orígenes de Java como lenguaje de programación se deben más al azar que al fin para el cual se utiliza hoy en día [Mohedano, 1998]. En 1990, Sun organiza un equipo de seis personas (entre las que se encontraban **James Gosling, Bill Joy** y **Patrick Naughton**), al que se denominó *Green*, para que desarrollasen algo innovador. Este grupo centra su atención en los dispositivos electrónicos de gran consumo, buscando una forma de comunicarlos y que pudieran ejecutar programas simples suministrados a través de una red. Dadas las características intrínsecas de estos aparatos, que se resumen en heterogeneidad y limitaciones de memoria, se dedican a diseñar un nuevo lenguaje de programación orientado a objetos que James Gosling denominó **Oak**. Este lenguaje estuvo listo en el verano de 1991, justo el mismo verano en que el servicio WWW iniciaba su expansión por Internet. En octubre de 1991 el grupo había crecido y Sun decidió crear una compañía autónoma que se denominó *First Person*, con las vistas puestas en la televisión interactiva, pero no cuajó el intento. Durante 1993 y parte de 1994 el grupo desarrolló diferentes proyectos en Oak, a la vez que buscaba a quién venderle esta tecnología, no consiguiendo ningún contrato. A finales de 1994 el equipo se disuelve, aunque algunas personas del grupo prosiguen en el intento de aplicar la tecnología Oak a los sistemas de escritorio basados en red. La difusión que por 1994 experimentaba el mundo web, abrió una puerta de solución a la tecnología desarrollada, ya que la filosofía cliente-servidor se ajustaba perfectamente a lo que se había desarrollado. En junio de 1994 surge la idea de utilizar a Oak como un producto en sí mismo, distribuyéndolo a través de Internet, a la vez que se baraja la idea de crear un sistema operativo basado en el lenguaje, consolidando la independencia de plataforma. Sun apoya en el otoño de 1994 la idea de regalar el lenguaje Oak, pero antes de su lanzamiento se dan cuenta de que ya existe un lenguaje con ese nombre, y se lo cambian por Java, presentándose oficialmente en la revista **SunWorld** en mayo de 1995. A lo largo de 1999 ha aparecido la última revisión de Java, la especificación de *Java 1.2*, también conocida como **Java 2**. En [Naughton, 1996] se cuenta la historia de este lenguaje según uno de sus autores, Patrick Naughton.

Java es un lenguaje que en su sintaxis recuerda mucho a C++ (y por tanto a C), pero sus semejanzas con C++ no van mucho más allá [Joyner, 1999].

Uno de los puntos más interesantes de Java es que no sólo es un lenguaje de programación, es una plataforma que se compone de un *lenguaje de programación* de propósito general orientado a objetos; de *archivos de clase* encargados de almacenar los ***bytecodes***, resultado de compilar los programas java, facilitando la independencia de plataforma y la movilidad entre redes; de una *máquina virtual* que se encarga de ejecutar los programas Java; y una *interfaz de programación de aplicaciones*.

Existe una numerosa bibliografía sobre Java, entre los que dan una visión general de Java como lenguaje de programación se pueden citar [Naughton, 1996], [Arnold and Gosling, 1997], [SUN, 2000]. Una buena referencia de consulta avanzada de Java 1.2 es [Jaworski, 1998].

Existen otros muchos lenguajes de programación orientados a objetos, tales como ***Oberon*** [Wirth and Reiser, 1992] – diseñado por **Niklaus Wirth** como sucesor de Modula-2; ***Modula-3*** [Harbison, 1992] – es un proyecto de Digital, que parte de Modula-2; ***Self*** [Ungar et al., 1992] – no está basado en clases sino en prototipos, teniendo en la herencia una relación entre objetos más que entre tipos; ***Ada 95*** [Ada95-Web] – es el resultado de añadir capacidades de Orientación a Objetos al lenguaje Ada 83; cuyo repaso sobrepasa el cometido de este apartado. Existen diversos fuentes bibliográficas que repasan y comparan diversos lenguajes orientados a objetos, entre ellos cabe destacar el capítulo 15 de [Rumbaugh et al., 1991], el apéndice A de [Booch, 1994], el capítulo 32 de [Meyer, 1997] o [Rans, 1999].

La Figura 4.6 ilustra la evolución de algunos de los principales lenguajes de programación orientados a objetos.

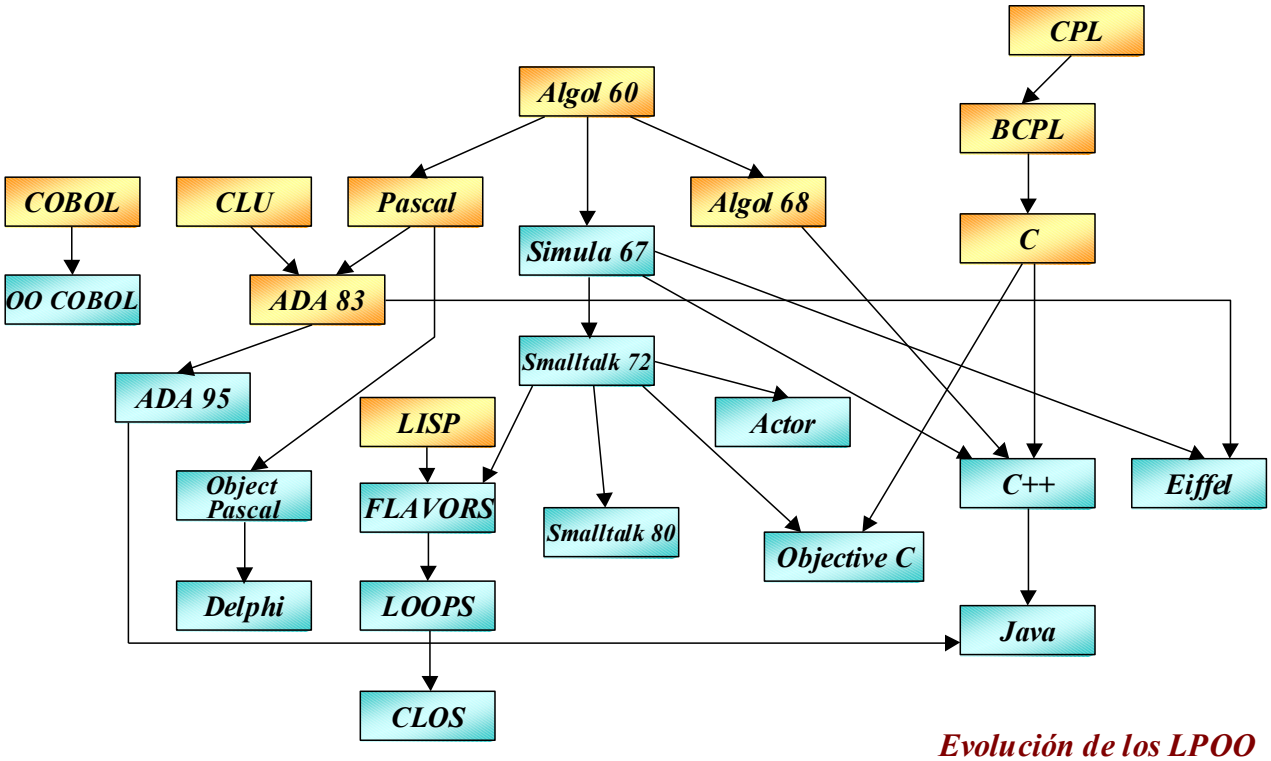

Figura 4.6. Evolución de los lenguajes de programación orientados a objetos

4.3.2.3 Evolución de los métodos orientados a objetos

Al igual que sucede en el paradigma estructurado, la tecnología de objetos debe dar cobertura a todo el ciclo de desarrollo de los sistemas software, es decir, al análisis, al

diseño y a la implementación; aunque se debe resaltar que estas fases se solapan y presentan unas formas más difuminadas que en el desarrollo estructurado.

Cuando a mediados de la década de los ochenta resurge la Orientación a Objetos, comienzan a surgir multitud de propuestas metodológicas con una orientación objetual, sólo en el período comprendido entre 1989 y 1994 el número de lenguajes de modelado orientados a objetos pasa de 10 a más de 50. Precisamente esta diversidad de métodos y notaciones ha sido una de las mayores barreras con que se encontraron las empresas y los departamentos de informática para la adopción de la Orientación a Objetos, conociéndose a este fenómeno como la *guerra de los métodos* [García y Pardo, 1998].

La cantidad de métodos que surgen en esta primera *hornada* se denominan metodologías o métodos de primera generación entre los que cabe destacar **Synthesis** [Bailin, 1989]**, RDD** (*Responsibility Driven Design*) [Wirfs-Brock et al., 1990], **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991], **OOD** (*Object Oriented Design*) [Booch, 1991], **OOSE** (*Object-Oriented Software Engineering*) – **Objectory** [Jacobson et al., 1993], o la metodología de **Shlaer y Mellor** [Shlaer and Mellor, 1992].

Hacia la mitad de década de los noventa aparecen en escena las nuevas versiones de los métodos más consolidados, así como nuevas incorporaciones que surgen como una acumulación de las características más destacadas de los métodos tradicionales junto con algunos añadidos: las metodologías o métodos de segunda generación. Pertenecientes a esta generación se tiene, entre otros, a **OMT-2** [Rumbaugh, 1996], **OOram** [Reenskaug et al., 1996], **OOA/D** (*Object-Oriented Análisis and Design*) [Booch, 1994], **Fusion** [Coleman et al., 1994]**, Moses** (Methodology for Object-Oriented Software Engineering of Systems) [Henderson-Sellers and Edwards, 1994a], [Henderson-Sellers and Edwards, 1994b]**, Syntropy** [Cook and Daniels, 1994] o **Medea** [Piattini, 1994].

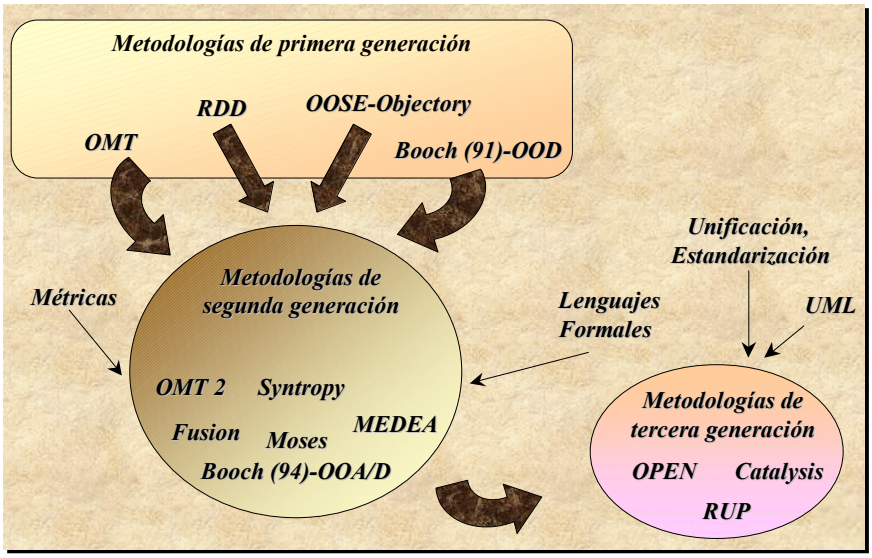

Figura 4.7. Generaciones de las metodologías orientadas a objetos

Ante esta situación de diversidad, en la parte final de la década de los noventa se empiezan a escuchar los primeros rumores sobre la necesidad de una unificación y una estandarización, esto ha dado lugar a la tercera generación de metodologías, de entre las que destacan **RUP** (*Rational Unified Process*) [Jacobson et al., 1999], **OPEN** (*Object-Oriented Process, Environment and Notation*) [Graham et al., 1997], [Henderson-Sellers et al., 1998], [Firesmith et al., 1998] o **Catalysis** [D'Souza and Wills, 1999].

En la Figura 4.7 se presenta un esquema de la evolución sufrida por las metodologías y métodos de diseño orientados al objeto.

El catalizador principal de esta unificación ha sido sin lugar la aparición en escena del lenguaje de modelado **UML** (*Unified Modeling Language*) de ***Rational Software Corporation***, que fue adoptado el 17 de noviembre de 1997 como lenguaje de modelado estándar por el **OMG** (*Object Management Group*) en su versión 1.1 [Rational et al., 1997]. OMG ha propuesto la especificación de UML para su estandarización internacional por parte de **ISO** (*International Organization for Standardization*).

UML se desarrolla en el seno de ***Rational Software Corporation*** con el apoyo de diversos colaboradores a lo largo de su historia, convirtiéndose en el elemento unificador de los lenguajes de modelado de los métodos OOA/D de **Grady Booch** [Booch, 94], OMT-2 de **James Rumbaugh** [Rumbaugh, 1996] y OOSE de **Ivar Jacobson** [Jacobson et al., 1993].

Grady Booch, director científico de Rational Software Corporation desde prácticamente su creación en 1980, empieza a planificar su estrategia a favor de la unificación de métodos. Así, para la comunidad de los métodos orientados al objeto la gran noticia en el OOPSLA'94 fue que James Rumbaugh había abandonado General Electric para unirse a Grady Booch en Rational Software Corporation para fusionar sus métodos. De esta manera el desarrollo de UML comienza en octubre de 1994, cuando Booch y Rumbaugh empiezan a trabajar para unificar los dos métodos que más repercusión habían alcanzado en la escena del ADOO, OOA/D y OMT. Como primer fruto de esta colaboración aparece en octubre de 1995 la primera versión pública de la descripción de la unión de sus métodos. Esta versión se presenta en el OOPSLA'95 con el nombre de Método Unificado versión 0.8 [Booch and Rumbaugh, 1995].

El siguiente paso en el proceso de unificación se produce a finales de 1995 cuando Rational compra a la empresa Objectory, y su fundador, el prestigioso Ivar Jacobson, se une a Rational como vicepresidente de Ingeniería de Negocio para acoplar su metodología OOSE al método unificado de Booch y Rumbaugh. Tras la incorporación de Jacobson a Rational, se les conoce a Booch, Rumbaugh y Jacobson por *los tres amigos*.

El primer paso que se da después de la incorporación de Jacobson es la creación de un lenguaje de modelado unificado debido principalmente a dos razones. En primer lugar el lenguaje de modelado permite que los tres métodos puedan evolucionar hacia

un punto común de forma independiente. Por otro lado, la unificación de la semántica y de la notación les permite colocarse en un lugar de privilegio en el mercado de los métodos orientados al objeto, debe tenerse en cuenta que la mayoría de la gente que dice utilizar un método orientado al objeto realmente lo que usa es la notación asociada a dicho método, olvidándose de los procesos asociados al método, así pues, contar con una notación gráfica universal era fundamental para la unificación de sus métodos, y de hecho la notación de UML se convierte en el estándar de facto de las notaciones en tecnología de objetos antes de su aceptación como estándar oficial.

La primera versión de este lenguaje de modelado aparece en junio de 1996 con el nombre de UML 0.9 [Booch et al., 1996a]. En septiembre de este mismo año aparece la versión 0.91 de UML [Booch et al., 1996b].

Durante 1996 invitan a otros expertos a colaborar con ellos, entre los colaboradores se encuentran nombres muy ilustres y de reconocido prestigio dentro de la comunidad de la Ingeniería del Software y más particularmente de la Orientación al Objeto. Citarlos a todos sería largo, pero como muestra se pueden mencionar a personajes de tanto renombre como *Peter Coad, Derek Coleman, Ward Cunningham, David Ambly, Eric Gamma, David Harel, Richard Helm, Ralph Johnson, Stephen Mellor, Bertrand Meyer, Jim Odell, Kenny Rubin, sally Shlaer, JohnVlissides, Paul Ward, Rebbeca Wirfs-Brock, Edward Yourdon* entre otros.

Además, se crea **OTUG** (*Object Technology Users Group*) como foro de discusión; una lista de correo en la que se discute y se comparten ideas sobre la tecnología de objetos, teniendo como trasfondo a UML.

Tras la salida a la luz de UML la comunidad de ADOO queda divida en dos grupos principales, aquellos que se unen a la estela de UML y aquellos que forman una coalición anti UML. Esta situación la aprovecha OMG para en 1996 crear el **OOAD SIG** (*Object-Oriented Analysis and Design Special Interest Group*) que se encargara de canalizar los esfuerzos para conseguir un estándar. De esta forma se organiza la ***OMG Task Force RFP*** (*Request For Proposal*), es decir una petición de propuestas para la creación de un estándar del lenguaje de modelado para los métodos de ADOO. Esta petición de propuestas sugería un lenguaje de modelado que contara con un metamodelo, una notación, una sintaxis y una semántica.

Rational va a responder a esta petición de propuesta con la versión 1.0 de UML en enero de 1997 [Booch et al., 1997a], [Booch et al, 1997b]. Esta versión está avalada por un conjunto de empresas de mucho prestigio dentro del mundo de la informática: *Digital Equipment Corporation, HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, y Unisys*.

Como era de esperar, la respuesta de Rational no fue la única, OMG recibió en enero de 1997 las respuestas de IBM & ObjecTime [Cook et al., 1997], Platinum Technology [Rivas et al., 1997], Ptech, Taskon & Reich Technologies y Softeam.

Durante los primeros meses de 1997 se hicieron predicciones de todo tipo y para todos los gustos. La notación de UML se había convertido en un estándar de facto apoyado por empresas de especial relevancia en el sector informático, lo cual colocaba a UML en una posición de privilegio, pero su oscura semántica y su metamodelo parecían no estar a la altura de otras propuestas, lo cual hacía peligrar su adopción como estándar por OMG en detrimento del resto de las propuestas. Así todo apuntaba a dos posibles soluciones. La primera de ellas, y quizás la peor, daba como resultado dos estándares: UML de Rational y OML de Platinum Technology. La segunda de ella apuntaba por una solución mixta que contemplase aspectos de todas las propuestas, siendo la que se llevó a cabo; así se unen al equipo liderado por Rational el resto de los equipos que enviaron propuestas, dando lugar a la versión 1.1 de UML [Rational et al., 1997], que fue enviada a OMG el 1 de septiembre de 1997, contribuyendo todos con sus ideas a la generación de una respuesta única.

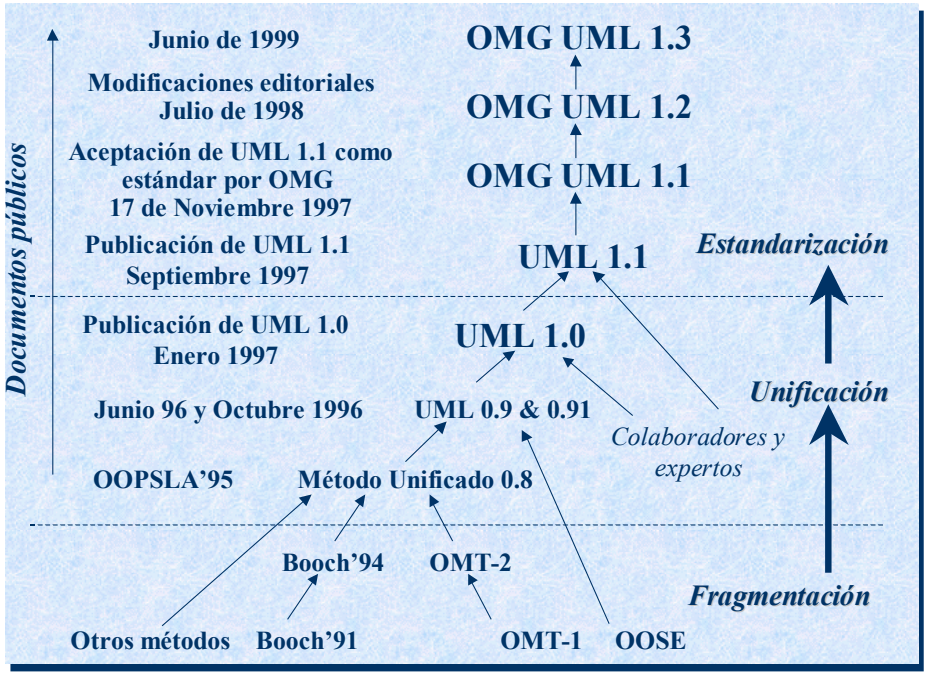

Figura 4.8. Evolución de UML

Esta propuesta es aceptada por OMG como estándar de lenguaje de modelado el 17 de noviembre de 1997, pasándose a denominar al lenguaje de modelado **OMG UML 1.1**. En julio de 1998 aparece la versión **OMG UML 1.2** [OMG, 1998], presentando sólo cambios editoriales. Casi un año más tarde, en junio de 1999 aparece **OMG UML 1.3** [OMG, 1999] con algunos cambios significativos, especialmente en lo tocante a la semántica, siendo ésta la versión en vigor a fecha de hoy[22].

[22] En la actualidad se está trabajando en una serie de cambios menores, que se verán reflejados en una versión 1.4 de UML (UML Revisión Task Force 1.4) que aparecerá a lo largo del año 2000, y en una revisión más profunda que aparecerá como UML 2.0. para el 2001 [Kobryn, 1999]. Una mayor información sobre estos trabajos se puede encontrar en la página web de UML Revison Task Force (http://uml.shl.com).

El camino seguido hasta la versión actual de OMG UML se puede resumir en la Figura 4.8.

Las referencias básicas de UML las constituyen los libros escritos por los autores principales del mismo [Rumbaugh et al., 1999], [Booch et al., 1999] y [Jacobson et al., 1999], donde los dos primeros están dedicados por completo al lenguaje de modelado, mientras que el tercero se dedica al proceso unificado.

4.3.3 Marco conceptual de la Orientación a Objetos

Los sistemas y el pensamiento orientado a objetos se asientan sobre una colección de conceptos que, en su mayor parte, aparecen en diferentes campos y desarrollos de las Ciencias de la Informática con anterioridad a que la Orientación a Objetos fuera un paradigma aceptado como tal.

Desde que los primeros principios de la Orientación a Objeto fueran establecidos en el campo de los lenguajes de programación, fundamentalmente con Simula y Smalltalk, una gran diversidad de campos de la computación han utilizado la tecnología de objetos en el desarrollo de sus teorías; esto ha provocado la existencia de un número de polisemias y sinónimos en relación con el conjunto central de los conceptos inherentes a la Orientación a Objetos. En [Wirfs-Brock and Johnson, 1990] se hace un esfuerzo por presentar los conceptos básicos del paradigma, clarificando su significado con respecto a diferentes interpretaciones.

Para solucionar esta confusión conceptual se establece un *idioma común* que permita la comunicación entre equipos de desarrollo de software con tecnología de objetos. El conjunto de conceptos y propiedades, que configuran este *idioma común* es lo que se conoce como **Modelo Objeto**, y que puede definirse como sigue:

> **"Un Modelo Objeto es un marco de referencia conceptual, en el que se establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los sistemas software soportados por la tecnología orientada al objeto. Este conjunto básico de conceptos deberá incluir como mínimo, los de abstracción, encapsulación, jerarquía y modularidad; y deberá considerar el sistema de información como un conjunto de entidades conceptuales modeladas como objetos e interactuando entre ellas"**
>
> **[Marqués, 1995]**

Desde la perspectiva de la evolución de los modelos objetos, puede considerarse que el primero de ellos, reconocido como tal, es el propuesto por **Anita K. Jones** [Jones, 1987] en el año 1978, definiendo el modelo objeto como un concepto y una herramienta, que proporciona las líneas directrices para caracterizar las entidades abstractas, en los términos en los que se conciben.

A partir de esta primera propuesta formal de modelo objeto, y ante la ausencia de un modelo común de referencia y de estándares universalmente aceptados, para estos

modelos, se ha asistido a una proliferación de propuestas de modelos objetos [Sernadas et al., 1989], [Wand, 1989], [Castellanos et al., 1991], [Bertino and Martino, 1993], [Marqués, 1995] entre otros. Además, se puede constatar que la mayoría de los trabajos en la Orientación a Objetos definen de forma implícita o explícita su propio modelo objeto, como por ejemplo el de UML [OMG, 1999] o el de OPEN [Firesmith et al., 1998].

Según **Grady Booch** [Booch, 1994] todo modelo objeto está formado por cuatro elementos fundamentales: *abstracción, encapsulamiento, modularidad* y *jerarquía*; y por tres elementos secundarios: *tipos, concurrencia* y *persistencia*. "*Fundamentales*" significa que un modelo que carezca de cualquiera de estos elementos no es orientado a objetos. Con "*secundarios*" quiere decirse que cada uno de ellos es una parte útil del modelo objeto, pero no esencial.

La *abstracción* se utiliza para combatir la complejidad. Surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias. Una abstracción se centra en la visión externa de un objeto, sirviendo para separar el comportamiento esencial de un objeto de su implantación.

Se puede definir abstracción como:

> **"Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes"**
>
> **Ian Graham, [Graham, 1994]**

> **"Las características esenciales que distinguen a una entidad de todas las demás entidades. Una abstracción define una delimitación relativa a la perspectiva del observador"**
>
> **[OMG, 1999]**

Mientras que la abstracción ayuda a las personas a pensar sobre lo que están haciendo, el *encapsulamiento* permite que los cambios realizados en los programas sean fiables con el menor esfuerzo. El encapsulamiento genera una cápsula, una barrera conceptual sobre la información y servicios de un objeto, haciendo que estos permanezcan juntos. El encapsulamiento es un medio para conseguir el principio de ocultación de la información [Parnas, 1972].

Se puede definir encapsulamiento como:

> **"Un principio de estado que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto"**
>
> **[Piattini et al., 1996]**

> **"Técnica de modelado e implementación que separa los aspectos externos de un objeto de los internos, detalles de implementación de un objeto"**
>
> **[Rumbaugh et al., 1991]**

La *modularidad* permite la fragmentación de un programa en componentes individuales, lo que puede reducir la complejidad en algún grado. Dicha fragmentación crea una serie de fronteras bien definidas y documentadas dentro del programa. Estas interfaces tienen una importancia incalculable para la comprensión del programa.

La modularidad se puede definir como:

> **"La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados"**
>
> **Grady Booch, [Booch, 1994].**

La abstracción es un concepto sumamente útil, pero demasiado extenso, excepto para los casos triviales. El encapsulamiento y la modularidad son medios para manejar las abstracciones. Con frecuencia, un conjunto de abstracciones forma una jerarquía.

La jerarquía se puede definir como:

> **"La jerarquía es una clasificación u ordenación de abstracciones"**
>
> **Grady Booch, [Booch, 1994]**

Las dos jerarquías principales son la jerarquía de clases, formada mediante relaciones de herencia entre las clases, y la jerarquía de partes, formada mediante relaciones de agregación.

Los objetos se comunican a través del *paso de mensajes*. Esto elimina la duplicación de datos, garantizando que no se propaguen los efectos de los cambios en las estructuras de datos encapsuladas dentro del objeto sobre otras partes del sistema. Así, un objeto accede a otro enviándole un mensaje, cuando esto ocurre, el receptor ejecuta el método correspondiente al mensaje. Un mensaje consiste en un nombre de un método más cualquier argumento adicional. El conjunto de mensajes a los que un objeto responde caracteriza su comportamiento.

Se define mensaje como:

> **"Llamada a una operación o a un objeto, en la que se incluye el nombre de la operación y una lista de valores de argumentos"**
>
> **[Rumbaugh et al., 1991]**

> **"Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento"**
>
> **[OMG, 1999]**

El concepto de tipo se deriva de las teorías sobre los tipos abstractos de datos. Se incluye el tipo como elemento separado de modelado de objetos porque el concepto de tipo pone énfasis en el significado de la abstracción en un sentido muy distinto.

Se puede definir *tipo* como:

> **"Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas"**
>
> **Grady Booch, [Booch, 1994]**

Un concepto muy ligado a la teoría de tipos es el *polimorfismo*, que indica que un solo nombre puede denotar objetos de muchas clases diferentes que se relacionan por una superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto de operaciones. Lo opuesto al polimorfismo es el *monomorfismo*, que se encuentra en todos los lenguajes con comprobación estricta de tipos y ligadura estática. Existe el polimorfismo denominado de *inclusión* cuando interactúan las características de la herencia y el enlace dinámico. Es una de las características más potentes de los lenguajes de programación orientados a objetos después de su capacidad para soportar la abstracción, y es lo que distingue a la programación basada en tipos abstractos de datos.

Se puede definir polimorfismo como:

> **"La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases"**
>
> **Ian Graham, [Graham, 1994]**

Un sistema que implique *concurrencia* puede tener muchos hilos de control (*algunos transitorios y otros permanentes durante toda la ejecución del sistema*). Mientras que la Programación Orientada a Objetos se centra en la abstracción de datos, encapsulamiento y la herencia, la concurrencia se centra en la abstracción de procesos y la sincronización; los objetos unifican los dos puntos de vista anteriores.

Se define la concurrencia como

> **"Concurrencia es la propiedad que distingue un objeto activo de uno que no está activo"**
>
> **Grady Booch, [Booch, 1994]**

Todo objeto software ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo. Así pues, se puede definir la *persistencia* como:

> **"La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)"**
>
> **Grady Booch, [Booch, 1994]**

4.3.3.1 La Orientación a Objetos en los cuerpos de conocimiento de la Ingeniería del Software

La Orientación a Objeto es un paradigma de desarrollo, y como tal se contempla dentro de la Ingeniería del Software, por lo tanto está presente en los cuerpos de conocimiento de la Ingeniería del Software.

En el SWEBOK propuesto por IEEE/ACM [Abran et al., 1999], [Abran et al., 2000] los métodos orientados a objetos aparecen en varias áreas de conocimiento:

- **Área de Construcción de Software:** Se realiza una taxonomía de los métodos de construcción de software, de forma que se distinguen métodos de construcción *lingüísticos, matemáticos* y *visuales*, donde cada uno de los cuales se estudia con respecto a cuatro principios (*reducción de la complejidad, anticipación de la diversidad, estructuración de la validación* y *uso de estándares externos*) [Bollinguer, 1999]. La Orientación al Objeto está presente de una forma relevante tanto en los métodos de construcción de software lingüísticos como en los visuales (Tabla 4.10).

	Reducción de la complejidad	**Anticipación de la diversidad**	**Estructuración de la validación**	**Uso de estándares externos**
Lingüísticos	• Plantillas • Patrones de diseño • Bibliotecas de componentes y *frameworks* • Encapsulación y tipos abstractos de datos	• Ocultación de la información • Autodocumen-tación • Conjuntos de métodos completos y suficientes • Herencia • Lenguajes "pegamento" para el uso de componentes	• Modularidad • Guías de estilo	• Lenguajes de programación estándares • Lenguajes de especificación de datos estándares (XML) • Estándares de comunicación entre procesos (COM, CORBA) • Software basado en componentes
Visuales	• Programación orientada a objetos • Programación visual	• Clases • Separación entre la interfaz gráfica y la funcionalidad	• Diseño completo y suficiente de los métodos de las clases	• Lenguajes de programación orientados a objetos estándares • Modelos de interfaces visuales estandarizados

Tabla 4.10. Presencia de la Orientación al Objeto en el área de Construcción de Software del SWEBOK de IEEE/ACM

- **Diseño del Software:** La Orientación a Objeto está presente en todos los temas que aborda esta área de conocimiento (*Conceptos básicos y principios, Calidad de diseño y métricas, Arquitectura software, Notaciones de diseño* y *Estrategias y métodos de diseño*) [Tremblay, 1999], como se recoge en la Tabla 4.11.

	Elementos del paradigma objetual
Conceptos básicos y principios	*Abstracción* *Cohesión y acoplamiento* *Separación de conceptos* *Encapsulación* *Ocultación de la información* *Interfaz vs implementación* *Modularidad* *Refinamiento*
Calidad de diseño y métricas	*Atributos de calidad* (reusabilidad, portabilidad...) *Métricas de diseño orientado a objetos*
Arquitectura software	**Estilos arquitectónicos:** *Abstracción de datos y organización OO* *Sistemas distribuidos (CORBA, DCOM)* **Descripciones arquitectónicas:** *Patrones* ***Frameworks* orientados al objeto**
Notaciones de diseño	**Diseño arquitectónico:** *Diagramas de clase* *CRC* *Diagramas de despliegue y de procesos* **Diseño detallado:** *Diagramas de colaboración* *Diagramas de secuencia* *Diagramas de transición de estados*
Estrategias y métodos de diseño	**Estrategias generales:** *Diseño Orientado a Objetos:* Método de Coad y Yourdon Fusion RUP Diseño por contrato Método de Shlaer y Mellor RDD

Tabla 4.11. Presencia de la Orientación al Objeto en el área de Diseño del Software del SWEBOK de IEEE/ACM

- **Infraestructura de la Ingeniería del Software:** La Orientación a Objetos aparece representada de nuevo en las tres subáreas de esta área de conocimiento: *Métodos de desarrollo, Herramientas software* e *Integración de componentes* [Carrington, 1999] (Tabla 4.12).

<table>
<tr><th>Subárea</th><th colspan="3">Elementos relacionados con la OO</th></tr>
<tr><td>Métodos de desarrollo</td><td colspan="3">Se clasifican en heurísticos, formales y de prototipado. Dentro de los primeros se hace mención explícita de los OO</td></tr>
<tr><td>Herramientas software</td><td colspan="3">Herramientas CASE que soportan el ciclo de desarrollo en la construcción de software OO</td></tr>
<tr><td rowspan="2">Integración de componentes</td><td>Definición de componentes</td><td>Modelos de referencia</td><td>Reutilización</td></tr>
<tr><td>Especificación de la interfaz
Especificación de protocolos
COTS</td><td>Patrones
Frameworks</td><td>Tipos de reutilización
Repositorios</td></tr>
</table>

Tabla 4.12. Presencia de la Orientación al Objeto en el área de Infraestructura de la Ingeniería Software del SWEBOK de IEEE/ACM

- **Área de Análisis de los Requisitos del Software:** Donde se hace mención explícita a las técnicas de Análisis Orientado a Objetos [Sawyer and Kotonya, 1999].
- **Área de Pruebas del Software:** Donde se hace referencia a las técnicas para realizar las pruebas a software construido bajo el paradigma orientado a objeto y software basado en componentes [Bertolino, 1999].

Por lo que respecta al **SWE-BOK** realizado para la **FAA** [Hilburn et al., 1999], también aparece incluida la Orientación a Objetos en diversos apartados como se muestra en la Tabla 4.13.

<table>
<tr><th>Categoría de Conocimiento</th><th>Área de Conocimiento</th><th>Unidad de Conocimiento</th></tr>
<tr><td>Fundamentos de Informática</td><td>Lenguajes de Programación</td><td>Paradigmas de programación
Donde se reconoce a la POO como un paradigma de programación</td></tr>
<tr><td rowspan="6">Ingeniería del Producto Software</td><td rowspan="2">Ingeniería de Requisitos del Software</td><td>Análisis de requisitos
Métodos de modelado OO</td></tr>
<tr><td>Especificación de requisitos
Lenguajes de modelado OO</td></tr>
<tr><td rowspan="2">Diseño del Software</td><td>Diseño arquitectónico
Técnicas de DOO</td></tr>
<tr><td>Especificación abstracta
Notaciones y técnicas de DOO</td></tr>
<tr><td rowspan="2">Codificación del Software</td><td>Implementación del código
POO</td></tr>
<tr><td>Reutilización de código
Reutilización vs OO</td></tr>
</table>

Tabla 4.13. Orientación a Objetos en el SWE-BOK

Por último, el **SE-BOK** propuesto por el **WGSEET** [Bagert et al., 1999] al no describir con tanto detalle los componentes de conocimiento de cada una de las áreas de conocimiento propuestas, no puede establecer a priori la presencia de la Orientación a Objetos en este cuerpo de conocimiento. Sin embargo, como este cuerpo de conocimiento viene acompañado de una propuesta curricular, con la descripción de los cursos, puede verse como la Orientación a Objetos tiene un papel destacado en lo que se denomina el **área central**, especialmente en los componentes de conocimiento *requisitos del software, diseño del software* y *construcción del software*.

4.3.4 La enseñanza de la Programación Orientada a Objetos

La Orientación a Objetos es en sí misma una sub-disciplina de la Ingeniería del Software, que requiere que sus conceptos básicos queden asentados de forma sólida para su correcta aplicación en el desarrollo de sistemas software, donde se quiere obtener ventaja de todo su potencial [D'Souza, 1996].

Para la enseñanza de la Orientación a Objetos se aconseja seguir un modelo de aprendizaje constructivista, más que un modelo de aprendizaje objetivista [Hadjerrouit, 1999]. Esto es así porque en la aproximación objetivista se ve el proceso de aprendizaje como una *transmisión pasiva de conocimientos*, donde no se necesita ningún conocimiento previo; el resultado de este enfoque es que el alumno acaba sin una adecuada comprensión de la base conceptual del paradigma objetual, con malos hábitos de programación y con serios malentendidos sobre la tecnología de objetos. Por su parte la aproximación constructivista presenta el aprendizaje como un *proceso activo de construcción*, donde los alumnos construyen su conocimiento sobre la base de su conocimiento previo, siendo necesario probar lo que se conoce y evaluar si entra en conflicto con los conceptos que están adquiriendo.

Desde un punto de vista constructivo de la Orientación a Objetos, se requieren tres tipos de conocimientos: *los conceptos propios del modelo objeto, un lenguaje de programación orientado a objetos* donde plasmar los conceptos y *supuestos con problemas* de donde obtener el conocimiento específico del dominio [Hadjerrouit, 1999].

El enfoque constructivo, se asemeja a la estructura de los ciclos de vida del desarrollo de software en la tecnología de objetos, ya que es iterativo e incremental. Esto es así, porque el conocimiento y el entendimiento se va adquiriendo, ampliando y madurando en una serie de etapas [Meyer, 1996].

Otro aspecto a tener en cuenta es el momento en el que se introduce la Orientación a Objetos en el currículo del alumno. Hay defensores de introducir la Orientación a Objeto desde el primer momento en la formación de los titulados en Informática [Tewari and Friedman, 1992], [Decker and Hirshfield, 1994], [Adams, 1996],

[Woodman et al., 1996], [Hadjerrouit, 1999]; aunque siendo conscientes que el aprendizaje de los conceptos propios de la tecnología de objetos es difícil para los nuevos alumnos porque requiere una forma de pensar diferente y más profunda en términos de computación [Hadjerrouit, 1999].

Sobre los conocimientos que se requieren para iniciarse en la Orientación a Objetos, hay quien opina que aquéllos que tienen experiencia en el desarrollo de sistemas software, típicamente en el paradigma basado en procedimientos, aprenden fácilmente la sintaxis de los lenguajes orientados a objetos, pero esto no significa que hayan captado la esencia del diseño orientado a objetos y sean capaces de plasmar dichos diseños con los lenguajes aprendidos [Rosson and Carroll, 1996]; de hecho hay estudios que indican que el conocimiento de lenguajes de tipo procedimental puede llegar ser una barrera conceptual para introducirse plenamente en la Orientación a Objetos [Hadjerrouit, 1999].

La forma más extendida para introducir la tecnología de objetos en los currículos de Informática es mediante los cursos de introducción a la programación, donde los lenguajes procedimentales y los métodos de diseño estructurado se verían desplazados por sus homónimos orientados al objeto.

Por supuesto, hay voces que no están de acuerdo con que los conceptos del paradigma de la Orientación a Objetos se introduzcan por la programación, defendiendo que los principios del modelo de objetos para realizar análisis y diseño deben cubrirse con gran detalle antes de llegar a su implementación real en un lenguaje de programación [Northrop, 1992]; o quién no está de acuerdo con que la orientación a objeto se centre sólo en la programación, defendiendo un enfoque más amplio que cubra todo el ciclo vital, con una perspectiva de Ingeniería del Software [Bézivin et al., 1992].

En la Universidad española hay muy pocos casos en los que la Orientación a Objeto aparezca en el primer curso de los planes de estudio de las Ingenierías Técnicas y Superiores en Informática. En su lugar la introducción de la tecnología de objetos se enfoca desde dos perspectivas, complementarias, a saber:

- Inmersos dentro de alguna asignatura de Ingeniería del Software, donde principalmente se estudian las características de los ciclos de vida que pueden seguir los desarrollos de sistemas software bajo el paradigma objetual, así como alguna propuesta metodológica orientada a objetos (*haciendo más hincapié en la parte del lenguaje de modelado, UML típicamente, que en la parte de proceso*).
- Como componentes centrales de asignaturas de programación avanzada, normalmente de carácter optativo, que se centran en técnicas de diseño y programación orientadas a objetos.

El primer enfoque cae dentro de la enseñanza de la Ingeniería del Software [Miller and Mingins, 1998], tal y como se presentó en el apartado 4.2.4 de este capítulo, siendo

una aproximación de carácter más general y abstracto que la que se da en el segundo enfoque, cuando el centro de atención se pone en los modelos del dominio de la solución.

En el contexto de la Universidad de Salamanca se sigue la aproximación más generalizada en España, esto es, en la Ingeniería Técnica en Informática de Sistemas, la Orientación a Objeto se reparte en las dos asignaturas que marcan el perfil de la plaza a concurso: *Ingeniería del Software* y *Programación Orientada a Objetos*.

En la primera de ellas, principalmente, se introducen los conceptos del modelo objeto, para posteriormente hacer una somera introducción del análisis y diseño orientado a objetos a la vez que se presentan los principios básicos de UML. En la segunda de ellas se repasan los conceptos del modelo objeto, pero con una mayor profundidad, haciendo hincapié en las decisiones de diseño y se utiliza un lenguaje de programación como medio (no como fin) para que el alumno asimile el paradigma objetual de una forma constructiva.

La asignatura de Programación Orientada a Objetos se presta a la utilización de los patrones pedagógicos (*reusable pedagogical design patterns*) [Lilly, 1996], [Proto-Patterns, 1999], [Bergin, 1998a], [Bergin, 1998b]. En concreto el proyecto de patrones pedagógicos [Proto-Patterns, 1999] recoge prácticas efectivas de docentes en tecnología de objetos. Estos patrones deben ser fáciles de repetir y de adaptar. Cada patrón debe ser descrito de manera que sea fácilmente instanciable para diferentes *lecciones* y por diferentes *educadores*. Muchos de estos patrones se refieren de forma explícita a temas de tecnologías de objetos, como por ejemplo [Bellin, 1999], [Prieto and Victory, 1999] o [Vaitkevitchius, 1999], aunque otros muchos se pueden aplicar a la docencia de cualquier asignatura, como por ejemplo [Bergin, 1998c], [Jalloul, 1999] o [Manns, 1999]. Los antipatrones son otra herramienta que se puede aplicar en la pedagogía [Dodani, 1999].

4.4 Revisión de los currículos internacionales

Se puede definir currículo como "*un plan para educar estudiantes, ofreciéndoles las características y el conocimiento necesarios para vivir y practicar competentemente una profesión. El currículo debe anticiparse al mundo cambiante en que los alumnos graduados vivirán y trabajarán*" [Denning, 1992].

En este apartado se analizan las propuestas más importantes en currículos internacionales en Informática, realizadas por instituciones de alto prestigio dentro del mundo informático. Para ello se han identificado tres categorías de programas que se estudiarán por separado: *Ciencia de la Informática (CS), Sistemas de Información (IS) o Gestión de Sistemas de Información (MIS)* e *Ingeniería del Software (SE)*.

Cada una de estas áreas presenta su propio enfoque, pero claramente tienen un núcleo de conocimiento común [Parrish et al., 1998].

La revisión que aquí se hace se centra en destacar la presencia de las materias objeto de este Proyecto Docente en las diferentes propuestas curriculares.

4.4.1 Currículos centrados en la Ciencia de la Informática

La educación en *Ciencia de la Informática e Ingeniería* ha sido un área activa durante toda la historia de la disciplina. En particular desde el establecimiento de los primeros Departamentos de Ciencias de la Computación a mediados de la década de los sesenta, se puso una especial atención al reto de educar a los alumnos en un campo tan sumamente cambiante y que evoluciona con tanta rapidez como es la Informática [Tucker et al., 1996].

Las propuestas curriculares más relevantes que aparecen en el campo de la Ciencia de la Informática o Ciencia de la Computación tienen como protagonistas a las dos asociaciones más prestigiosas en el mundo informático, ACM y IEEE-CS. Estas organizaciones publican por separado diferentes propuestas curriculares entre los años 1968 y 1983 (*la evolución de los currículos en la Ciencia de la Informática se detalla en el Cuadro 4.5*), para terminar aunando esfuerzos para definir una propuesta curricular única en el campo de la Ciencia de la Informática e Ingeniería, la propuesta de ACM/IEEE-CS de 1991 (también conocida como *Computing Curricula 1991*) [Tucker et al., 1991a] convirtiéndose en una referencia internacionalmente aceptada para el establecimiento de los Planes de Estudio de las titulaciones de Informática en la Universidad, incluyendo a la Universidad en España.

Aunque ACM parte de una tradición más proclive a la Ciencia de la Informática, en la última década del siglo XX se ha acercado a la parte de Ingeniería, convergiendo con IEEE-CS, de orientación tradicionalmente ingenieril. De hecho, ACM se está abriendo incluso al mundo de los Sistemas de Información, cosa que nunca ha hecho IEEE.

En 1968, ACM publica el primer currículo informático, denominado *Currículo 68* [ACM, 1968], donde sus propios autores manifiestan que iba destinado a los que pretendían dedicarse a la investigación dentro de la Informática.

En 1978, ACM revisa su propuesta curricular [Austing et al., 1979], de forma que este informe, con sólo unas pequeñas modificaciones [Koffman et al., 1984], [Koffman et al., 1985], constituye un estándar para la educación en Informática, hasta 1991.

En 1983, IEEE-CS publica un currículo independiente para Ciencia de la Informática e Ingeniería [Education Activities Board, 1983].

En 1986 se presenta un currículo para las escuelas de *artes liberales* [Gibbs and Tucker, 1986].

Dado que en los trabajos publicados por las dos principales sociedades de Informática tenían bastantes partes en común, deciden aunar esfuerzos y elaborar una única propuesta avalada por ambas sociedades. Así, en 1985 se crea un grupo de trabajo dirigido por **Peter Denning** y formado por miembros de ACM e IEEE-CS; este grupo publica un informe en diciembre de 1988 [Denning et al., 1988]. Se decide continuar con el trabajo para desarrollar las recomendaciones para un currículo completo, para lo que se crea otro grupo de trabajo conjunto en febrero de 1988 (*The Joint ACM/IEEE-CS Curriculum Task Force*), que da lugar al Computing Curricula 1991 [Tucker et al., 1991a].

A finales de 1998, la **ACM Education Board** y la **Educational Activities Board de IEEE-CS** establecieron un nuevo grupo de trabajo para preparar el *Curriculum 2001*.

Cuadro 4.5. Evolución de las propuestas curriculares en CS

La propuesta ACM/IEEE-CS de 1991 sigue actualmente en vigor, esperándose una nueva propuesta para el año 2001, Currículo 2001 [Roberts et al., 1999], [Chang et al., 1999], y será objeto de un estudio pormenorizado en el siguiente subapartado.

4.4.1.1 La propuesta conjunta ACM/IEEE-CS (Curricula 91)

La recomendación conjunta (ACM/IEEE-CS 91) [Tucker et al., 1991a], [Tucker et al., 1991b] incluye un currículo general en Informática, adaptable a las necesidades concretas de cada institución que imparta titulaciones como "*Computer Science*", "*Computer Engineering*", "*Computer Science and Engineering*" y otras similares; pretende ser flexible y adaptarse a los nuevos cambios tecnológicos que se vayan produciendo. El informe constituye un conjunto de guías, más que una prescripción de cursos concretos, en una titulación universitaria de Informática. El contenido de este informe puede resumirse en los siguientes puntos:

- Una identificación de objetivos de un plan de estudios universitario en la disciplina de Informática.
- Una definición de qué es la Informática como disciplina vista en forma bidimensional "área/proceso", que comprende la definición de nueve áreas temáticas y tres procesos, uno para cada una de las áreas (*Teoría, Abstracción* y *Diseño*).
- Una colección de material curricular avanzado y suplementario que posibilita la profundización en el estudio en algunas de las nueve áreas identificadas. El desarrollo de estos materiales variará en función de los intereses y posibilidades de cada institución que los imparta.
- Un conjunto de consideraciones pedagógicas y curriculares que gobiernen la materialización de los anteriores requisitos comunes y materiales avanzados/suplementarios en una titulación universitaria completa. Se incluyen aspectos como *el papel de los laboratorios, la programación, las matemáticas, conceptos éticos, sociales y profesionales* y hace explícita la noción de los "**conceptos recurrentes**" comunes a diferentes áreas, independientes de una tecnología en particular y repetidas por toda la disciplina informática.

A lo largo del documento se insiste en que un currículo es algo más que un conjunto de cursos y por eso tiene que conocerse no solamente la materia básica, sino también tienen que comprenderse los tres procesos o puntos de vista: teoría, abstracción y diseño.

Influencias sobre el *Computig Curricula 91*

Los siguientes puntos resumen los aspectos más significativos de algunos de los trabajos anteriores que han tenido influencia sobre ACM/IEEE-CS91:

- El informe de **ACM de 1968** [ACM, 1968] supone una de las primeras tentativas de definir el ámbito de la Informática como disciplina. Además de incluir un currículo, define los principales campos de la Informática; concretamente identifica tres áreas: *estructuras de información y procesos, sistemas de proceso de información* y ***metodologías***. Propone un currículo central compuesto de cuatro cursos básicos (*algoritmos y programación, estructura de computadores y sistemas, estructuras discretas* y *cálculo numérico*) y de cuatro cursos más avanzados (*estructuras de datos, lenguajes de programación, organización de computadores* y *programación de sistemas*). Estos cursos enfatizaban el análisis numérico y el hardware, omitiendo la Ingeniería del Software [Tucker and Wegner, 1994].
- El informe de **ACM de 1978** [Austing et al., 1979], proporcionaba descripciones detalladas de cursos universitarios en Informática, subrayando el término "*Computer Science*" (al igual que el anterior de 1968) y la importancia de la programación. La visión aquí es muy dependiente de la máquina, y solamente en uno de los cursos "*opcionales*" avanzados se hace referencia a un curso de "***Diseño y Desarrollo de Software***" que podría incluirse en el currículo; incluye técnicas "top-down" y estructuradas de diseño, formación de equipos de desarrollo y gestión de proyectos.
- El informe de **la Educational Activities Board de IEEE-CS de 1983**, describía una serie de áreas para Informática, considerando la orientación "*Computer Science and Engineering*". En este informe se hacían recomendaciones sobre material de laboratorio y sobre el propio laboratorio como soporte a las clases teóricas; utilizaba una estructura modular para organizar los temas y construir los cursos, proponiéndose como una base para configurar planes de estudios adaptados a distintas situaciones y centros particulares. En este informe se introduce la **Ingeniería del Software** en varios niveles: *como área temática básica*, *como Laboratorio específico de Ingeniería del Software* y *como área temática avanzada*, con un carácter dominado por las técnicas y métodos estructurados, propios de la época, pero subrayando ya la importancia de la fase de requisitos, modelado conceptual, prototipado y validación y verificación.
- El informe de 1989 "*Computing as a discipline*" [Denning et al., 1989], defiende la integración del trabajo de laboratorio con las clases teóricas, subrayando la importancia de introducir aspectos de *diseño* en el currículo y propone que en los cursos introductorios se dé una visión global de toda la carrera, incidiendo en los conceptos fundamentales, mientras que en los cursos superiores se debe ir profundizando en cada uno de esos temas. En este informe se proponen ya las nueve áreas temáticas que, posteriormente, aparecen en el documento ACM/IEEE-CS 91, distinguiendo para cada una

de ellas ***Teoría*** (*matemáticas subyacentes*), ***Abstracción*** (*modelado, análisis*) y ***Diseño*** (*especificar soluciones, estudiar alternativas*).

Objetivos del programa de graduación. Perfil de los graduados

El programa debe preparar a los estudiantes para la comprensión de las materias relacionadas con la computación en dos aspectos: *como una disciplina académica* y *como una profesión dentro de su contexto social*. Los estudiantes deben adquirir las habilidades para poder mantenerse actualizados y evaluar las ideas nuevas. Entre estas habilidades se incluyen el leer publicaciones, la asistencia a seminarios y la evaluación de su contenido, así como el trabajo en equipo en proyectos relacionados con problemas de actualidad.

En consecuencia, el primer objetivo del programa de formación ha de ser el proporcionar una cobertura básica, amplia y coherente de la disciplina de la computación. Los estudiantes deben comprender las relaciones existentes entre las diferentes áreas de la computación.

El programa debe preparar a los estudiantes para aplicar sus conocimientos a problemas específicos y con restricciones para elaborar soluciones. Esto incluye la *capacidad de definir un problema claramente*, *determinar su posibilidad de tratamiento, determinar la oportunidad de consultar con expertos, evaluar y elegir una estrategia de solución adecuada; estudiar, especificar, diseñar, implementar, probar, modificar y documentar esa solución, evaluar alternativas y realizar análisis de riesgos del diseño, integrar tecnologías alternativas en la solución y comunicar la solución tanto a los compañeros, a los profesionales de otros campos como al público en general*. Esto incluye la capacidad de trabajo en equipos durante todo el proceso de resolución de problemas.

El programa debe proporcionar la suficiente exposición del amplio cuerpo de teoría que subyace en el campo de la computación, de forma que los estudiantes aprecien la profundidad intelectual y los elementos abstractos que continuarán retando a los investigadores en el futuro.

Los graduados tienen que tener presente la alta tasa de cambio tecnológico, la tasa de crecimiento relativo en la teoría de la computación y la interacción delicada que tiene lugar entre las dos.

Principios subyacentes en el diseño curricular

En esta propuesta curricular se identifican nueve áreas de conocimientos y tres procesos que caracterizan las diferentes metodologías operativas utilizadas en la investigación y el desarrollo de la computación.

Cada una de las áreas identificadas tiene una base teórica significativa, abstracciones significativas y realizaciones significativas de diseño e implementación.

Las nueve áreas son: *Algoritmos y estructuras de datos; Arquitectura; Inteligencia artificial y robótica; Bases de datos y recuperación de la información; Comunicación hombre-máquina; Computación numérica y simbólica; Sistemas operativos; Lenguajes de Programación* y *Metodología e Ingeniería del Software*. De todas ellas las relacionadas con este Proyecto Docente son:

- ***Comunicación Hombre-Máquina***. El propósito principal de esta área es la transferencia eficiente de información entre el hombre y la máquina. Se incluyen gráficos, factores humanos que afectan a la interacción eficiente, y la organización y visualización de la información para una utilización efectiva por parte de las personas.
- ***Metodología e Ingeniería del Software***. El propósito principal de esta área es la especificación, el diseño y la producción de grandes sistemas software. El interés se centra en los principios de programación y del desarrollo del software, la verificación y la validación del software, y la especificación y producción de sistemas software que sean seguros y fiables.

La Informática es vista simultáneamente como una disciplina matemática, científica y de ingeniería. Los diferentes profesionales en cada una de las áreas emplean diferentes metodologías operativas, o procesos, en el curso de sus trabajos de investigación, desarrollo y aplicación.

El primero de estos procesos es el llamado **teoría**, está fundamentado en las matemáticas, y se utiliza en el desarrollo de teorías matemáticas coherentes. Los principales elementos que componen este proceso son: *definiciones y axiomas*, *teoremas*, *pruebas* e *interpretación de resultados*.

El segundo proceso, denominado **abstracción**, está enraizado en las ciencias experimentales, y consta de los siguientes elementos: *recolección de datos y formulación de hipótesis*, *modelado y predicción*, *diseño de un experimento*, *análisis de resultados*. Cuando las personas hacen abstracción están modelando algoritmos, estructuras de datos o arquitecturas, por ejemplo; prueban hipótesis acerca de esos modelos, toman decisiones de diseño alternativas... A los estudiantes hay que introducirles en la abstracción a través de las clases y de los laboratorios.

El tercer proceso, llamado **diseño**, está enraizado en la Ingeniería y se utiliza en el desarrollo de un sistema o dispositivo para resolver un determinado problema. Consta de las siguientes partes: *requisitos*, *especificaciones*, *diseño*, *implementación* y *pruebas*. Cuando los profesionales de la computación diseñan, han de implicarse en la conceptualización y la realización de sistemas en el contexto de las restricciones del mundo real. Los estudiantes aprenden diseño mediante la experiencia directa y mediante el estudio de los diseños de otros. Los proyectos de laboratorio se orientan hacia el diseño, proporcionando a los estudiantes una experiencia de primera mano en el desarrollo de un sistema o un componente de un sistema para resolver un problema

particular. Estos proyectos de laboratorio enfatizan en la síntesis de las soluciones prácticas a problemas y, por tanto, obligan a los estudiantes a evaluar las alternativas, costes y rendimientos en el contexto de las restricciones del mundo real. Los estudiantes desarrollan la capacidad de realizar estas evaluaciones viendo y discutiendo ejemplos de diseños, así como recibiendo información sobre sus propios diseños.

Conceptos recurrentes

Existen ciertos conceptos fundamentales que aparecen de forma recurrente en el diseño de los currículos de computación, y que representan un papel importante en el diseño de los cursos individuales. Estos conceptos son *ideas, materias, principios* y *procesos* que ayudan a unificar una disciplina académica en su substrato. En la propuesta curricular ACM/IEEE-CS91 se han identificado doce conceptos, a saber:

- **Ligadura**. El proceso de concretar abstracciones mediante la asociación de propiedades adicionales a dicha abstracción. Por ejemplo la asociación de procesos a procesadores o la creación de instancias concretas a partir de descripciones abstractas.
- **Complejidad de los grandes problemas**. Los efectos del crecimiento no lineal de la complejidad cuando crece el tamaño del problema.
- **Modelos conceptuales y formales**. Diferentes modos de formalizar, caracterizar, visualizar y concebir ideas o problemas. Los ejemplos incluyen modelos conceptuales del tipo de los tipos abstractos de datos y los modelos semánticos, y lenguajes visuales utilizados en la especificación y diseño de sistemas, tales como flujos de datos y diagramas entidad-relación.
- **Consistencia y compleción**. Incluye la consistencia de un conjunto de axiomas que sirven como especificación formal, la consistencia de la teoría con los hechos observados y la consistencia interna de un lenguaje. La compleción incluye la suficiencia de un conjunto de axiomas dados para capturar todos los comportamientos que se desea, la suficiencia funcional de los sistemas software y hardware, y la capacidad de un sistema para comportarse adecuadamente en condiciones de error o situaciones no previstas.
- **Eficiencia**. La medida de los costes relativos de los recursos tales como espacio, tiempo, dinero o personal.
- **Evolución**. El hecho del cambio y sus implicaciones.
- **Niveles de abstracción**. La naturaleza y uso de la abstracción en computación; la utilización de abstracciones para manejar la complejidad, estructurar sistemas, ocultar detalles y capturar patrones recurrentes. La capacidad de representar una entidad o sistema por abstracción tiene diferentes niveles de detalle y especificidad.

- **Ordenación en el espacio**. De los conceptos de localización y proximidad en la disciplina de la Informática. Además de la localización física, como en las redes o en la memoria de un computador, incluye el emplazamiento de la organización (por ejemplo, de procesadores y procesos; definiciones de tipo y las operaciones asociadas) y la localización conceptual (por ejemplo, el ámbito del software, la cohesión y el acoplamiento).
- **Ordenación en tiempo**. El concepto de tiempo en la ordenación de los eventos. Esto incluye el tiempo *como un parámetro en los modelos formales* (como por ejemplo en la lógica temporal), *como sinónimo de sincronización de procesos* o *como una parte esencial en la ejecución de los algoritmos*.
- **Reutilización**. La capacidad de una técnica, concepto o componente de sistema para ser reutilizado en un nuevo contexto o situación.
- **Seguridad**. La capacidad de los sistemas software y hardware para responder y defenderse de las peticiones no autorizadas, inapropiadas o imprevistas.
- **Decisiones y consecuencias**. El fenómeno de las decisiones en Informática y de las consecuencias que acarrean; los efectos técnicos, económicos, culturales que se derivan de la selección de una determinada alternativa de diseño.

El papel de los laboratorios

Un currículo de formación en Informática se compone, idealmente, de un programa integrado de clases teóricas y experiencias de laboratorio.

El papel de los laboratorios es muy importante en el diseño de un currículo en formación Informática, presentando las siguientes características y ventajas:

- Los laboratorios permiten demostrar la aplicación de los principios en el diseño, implantación y prueba de los sistemas software.
- Los laboratorios enfatizan las técnicas que utilizan las herramientas actuales y conducen hacia métodos experimentales adecuados incluyendo la presentación oral y escrita de los hallazgos.
- Los ejercicios de laboratorio han de estar diseñados para mejorar la experiencia del estudiante en las metodologías del software mediante el desarrollo de diseños e implantaciones.
- Las experiencias del laboratorio incrementan la capacidad de resolver problemas, las habilidades analíticas y la capacitación profesional.

Se diferencian dos tipos de laboratorios, *laboratorios abiertos* y *laboratorios cerrados*. Los primeros consisten en una asignación de trabajo que se llevará a cabo sin supervisión. Un laboratorio cerrado conlleva la asignación de trabajo de forma

planificada, estructurada y supervisada. La finalización de un trabajo de laboratorio debe de estar acompañada de un informe oral y/o escrito por parte del estudiante.

La unidad de conocimiento

El currículo en Informática se organiza sobre la base de unidades del conocimiento. Se entiende por unidad de conocimiento *la designación de una colección coherente de materias dentro de una de las nueve áreas principales de la disciplina de la computación*.

Para cada unidad de conocimiento se especifican los contenidos bajo el epígrafe de materias, los laboratorios propuestos, la posible relación con otras unidades de conocimiento, las unidades de conocimiento que son prerrequisito y si la unidad del conocimiento es requisito para otras unidades de conocimiento.

Las unidades de conocimiento que se definen en la propuesta curricular ACM/IEEE-CS91, relacionadas con el presente Proyecto Docente caen dentro del área de Metodología e Ingeniería del Software; cuyas unidades de conocimiento a su vez se recogen en la Tabla 4.14.

IS: Metodología e Ingeniería del Software *(se recomiendan unas 44 horas teóricas)*
IS1: Conceptos fundamentales para la resolución de problemas
IS2: El proceso de desarrollo del software
IS3: Requisitos y especificaciones del software
IS4: Diseño e implementación del software
IS5: Verificación y validación

Tabla 4.14. Unidades de conocimiento para el área de Metodología e Ingeniería del Software

De las unidades de conocimiento de esta área se describen aquéllas que se estima que deben estar contempladas en las asignaturas que se ajustan al perfil y al entorno en el que se va a desarrollar la docencia objeto de la plaza a concurso.

- **IS2.- El proceso de desarrollo del software.**

  Introducción a los modelos y elementos relacionados con el desarrollo de software de calidad. Utilización de entornos y herramientas que faciliten el diseño y la implantación de grandes sistemas software. El papel y la utilización de los estándares.

  1) Materias.

  a. Modelos del ciclo de vida del desarrollo del software.
  b. Objetivos del diseño del software.
  c. Documentación.
  d. Gestión y control de configuraciones.
  e. Elementos de la fiabilidad del software.

f. Mantenimiento.
g. Herramientas de especificación y diseño, herramientas de implementación.

2) Laboratorios (abiertos).

a. Implementar un prototipo para una especificación determinada.
b. Dado un diseño de software y una implementación intermedia de un desarrollo iterativo, completar la implementación correspondiente a la siguiente iteración.
c. Crítica de un conjunto de documentación determinado.
d. Dada una implementación, unas especificaciones y un conjunto de nuevas especificaciones, modificar el código de acuerdo a las nuevas especificaciones.

3) Relacionado con: Especificaciones y requisitos del software.

4) Prerrequisitos: Tipos abstractos de datos.

5) Requisito para: Diseño e implementación del software.

- **IS3.- Especificaciones y requisitos del software.**

Introducción al desarrollo de especificaciones formales e informales para la definición de los requisitos de un sistema software.

1) Materias.

a. Especificaciones informales.
b. Especificaciones formales, especificaciones algebraicas de los TAD, precondiciones y postcondiciones.

2) Laboratorios.

a. Producir un documento de análisis de requisitos.
b. Producir un documento con las especificaciones formales correspondientes a un conjunto de especificaciones informales.

3) Relacionado con: El proceso de desarrollo del software, control de tipos, semántica de los lenguajes.

4) Prerrequisitos: Tipos abstractos de datos.

5) Requisito para: Diseño e implementación del software, verificación y validación del software.

- **IS4.- Diseño e implementación del software.**

Introducción a los paradigmas principales que rigen el diseño y la implementación de grandes sistemas software.

1) Materias.

a. Diseño dirigido por funciones/procesos.
b. Diseño ascendente, soporte para reutilización.
c. Estrategias de implementación (por ejemplo, ascendente, descendente, equipos)
d. Elementos de la implementación, mejora de rendimientos, depuración.

2) Laboratorios (abiertos).

a. Realizar un diseño objeto para un conjunto de especificaciones.
b. Implementar el diseño anterior
c. Dado una especificación de problema y un conjunto de módulos ejecutables con sus especificaciones, realizar un diseño ascendente, reutilizando lo más posible.
d. Realizar una implementación descendente para un diseño software dado.

3) Relacionado con: Bases de datos, paradigmas de programación.

4) Prerrequisitos: IS2, IS3.

Otras consideraciones

La propuesta curricular ACM/IEEE-CS91 propone un curso optativo de nivel avanzado en Ingeniería del Software, centrado en los métodos y herramientas necesarios para incrementar la calidad, además de reducir el coste y la complejidad de mantenimiento de los sistemas software. Este curso tiene como prerrequisitos, además de diversas unidades de las áreas de algoritmos y estructuras de datos, cálculo numérico, lenguajes de programación y asuntos tanto éticos como sociales de la profesión, así como todo el área de metodología e Ingeniería del Software.

Se incluye una propuesta detallada de un programa universitario en Informática con énfasis en la Ingeniería del Software (*Implementation G: A Program in Computer Science – Software Engineering Emphasis*) [Tucker et al., 1991a].

En cuanto a la tecnología de objetos, cabe decir que tiene muy poca presencia en esta propuesta curricular (unas diez horas teóricas divididas en cuatro áreas [Osborne, 1992]), quizás porque ésta se centra en describir de una forma general el esqueleto básico de la Informática como disciplina, más que en establecer su cuerpo de conocimiento.

Aunque no es perfecta, esta propuesta curricular ha sido la que mayor influencia ha tenido en un plano internacional, incluyendo a España. Durante todos los años que lleva en vigor el *Computing Curricula 91*, se han propuesto diferentes modificaciones para incorporar o enfatizar diferentes aspectos, como la tecnología de objetos, la Ingeniería del Software o los Sistemas de Información; algunas de estas propuestas son: [Scragg et al., 1994], [Knight et al., 1994], [Shackelford and LeBlanc, 1994], [Hirmanpour et al., 1995], [Reynolds and Fox, 1996], [Pham, 1997] o [Jackson et al., 1997].

4.4.1.2 La propuesta conjunta ACM/IEEE-CS (Curricula 2001)

Reconocidas las limitaciones del *Computing Curricula 91*, de nuevo ACM e IEEE-CS se unen para crear un nuevo grupo de trabajo que defina una nueva propuesta curricular en el campo de la Ciencia de la Informática, que se espera esté terminada en el año 2001.

Actualmente este grupo de trabajo ha adoptado diez principios [Chang et al., 1999]:

1. La futura propuesta curricular debe perseguir el doble objetivo de formar buenos investigadores y buenos profesionales.
2. La Informática debe integrar la Matemática, la Ciencia y la Ingeniería.
3. Las unidades de conocimiento están disponibles en el proceso del diseño currículo. Estas unidades deben actualizarse para adaptarse a los nuevos conceptos que han aparecido en los últimos diez años.
4. La propuesta curricular debe ofrecer una guía para el diseño de cursos individuales. Se espera que en este sentido debe ser más efectivo que su antecesor.
5. El currículo 2001 debe identificar un conjunto relativamente pequeño de conceptos centrales y propiedades que son requeridos por todos los estudiantes.
6. El currículo 2001 debe ofrecer guías para cursos avanzados.
7. La propuesta curricular debe tener ámbito internacional.
8. La propuesta curricular debe ser el resultado de una participación significativa con la industria.
9. Debe tener en cuenta la práctica profesional.
10. Debe satisfacer las necesidades de los programas universitarios.

El grupo de trabajo ha identificado una serie de áreas de conocimiento clave, que deberán estar representadas en el informe final; éstas son: *matemáticas y ciencia, estructuras discretas, algoritmos y complejidad, arquitectura, sistemas inteligentes, gestión de la información, interacción hombre-máquina, gráficos por computador y visualización, sistemas operativos, fundamentos de programación, lenguajes y traducción,* ***Ingeniería del Software****, redes, ciencia computacional* y *asuntos sociales, éticos, legales y profesionales*.

4.4.1.3 Modelo de currículo para las artes liberales

Este modelo de currículo [Gibbs and Tucker, 1986] presenta una aproximación alternativa a la disciplina de la Informática, que pone un gran énfasis en que la Ciencia de la Informática tiene un cuerpo coherente de principios científicos. Define la Informática como un estudio sistemático de propiedades formales, implementación y aplicación de algoritmos y estructuras de datos.

La Ingeniería en esta propuesta curricular está ausente; en una posterior revisión [Walker and Schneider, 1996] se le asignan trece horas a la Ingeniería del Software (frente a las cuarenta y cuatro que recomendaba el *Computing Curricula 91*). Se presenta también un curso optativo que lleva por nombre *Ingeniería del Software*.

Existe alguna propuesta para la creación de un programa de estudios centrado en la Ingeniería del Software para su desarrollo en las Escuelas de Artes Liberales, como por ejemplo [Tymann et al., 1994].

4.4.2 Currículos centrados en los Sistemas de Información

Los Sistemas de Información es una parte esencial de las organizaciones. Son sistemas complejos que requieren tanto experiencia técnica como de organización para el diseño, el desarrollo y la gestión.

Hay una estrecha relación entre los Sistemas de Información y la Ciencia de la Informática y la Ingeniería del Software. Sin embargo, hay grandes diferencias ya que los Sistemas de Información se concentran en la parte organizativa y en la aplicación de las tecnologías de la información para sus objetivos.

Ninguna de las propuestas curriculares conjuntas de ACM/IEEE-CS entra en los Sistemas de Información, ya que son ajenos a la aplicación de las tecnologías de la información a los Sistemas de Información en las empresas u organizaciones; es decir, estarían cercanos a lo que en España se denomina *Ingeniería Técnica en Informática de Gestión*.

Dentro de un plan de estudios centrado en los Sistemas de Información, la Ingeniería del Software es una disciplina instrumental y su importancia relativa dependerá del grado de orientación tecnológica que se le quiera dar a la titulación, así hay titulaciones basadas en Sistemas de Información más orientadas al mundo empresarial, mientras otras están más orientadas al desarrollo de aplicaciones; estas últimas son las que utilizan conocimientos de Ingeniería del Software.

El desarrollo de currículos para Sistemas de Información comienza a principios de la década de los setenta, con el plan de estudios definido por la ACM [Ashenhurst, 1972]. Las razones de esta incursión en terrenos tradicionalmente ajenos fueron [Camps, 1999]:

a) El sector de los Sistemas de Información era y es el sector de aplicación de la Informática que más técnicos solicita y emplea.

b) En EEUU algunos departamentos de Ciencias de la Computación impartían enseñanzas propias de los Sistemas de Información.

La evolución histórica de los modelos de currículos para los Sistemas de Información se resume en el Cuadro 4.6.

La presencia de la Ingeniería del Software en estos modelos de currículos empieza a ser significativa a partir del modelo IS'90 definido por la **DPMA** (*Data Processing Management Association*) [Longenecker and Feinstein, 1991].

En el último modelo de currículo **IS'97** definido por la **ACM** (*Association for Computing Machinery*), la **AIS** (*Association for Information Systems*) y la **AITP** (*Association of Information Technology Professionals*) formalmente **DPMA** [Davis et al., 1997] presenta veinte subáreas significativas, de las que las áreas de **Análisis, diseño e implementación de Sistemas de Información** y de **Gestión de Proyectos** serían las más directamente relacionadas con la Ingeniería del Software, aunque muchas otras tendrían una relación más colateral. Por su parte el curso que más directamente se relaciona con la Ingeniería del Software es el que lleva el epígrafe **SI'97.7-Análisis y diseño lógico**, estando también relacionados los cursos con epígrafes **SI'97.8, SI'97.9** y **SI'97.10**; la descripción de estos cursos se encuentra en la Tabla 4.15**.**

	ÁMBITO	TEMAS
SI'97.7-Análisis y diseño lógico	Ofrece una comprensión del proceso de desarrollo y modificación de los sistemas software. Permite a los alumnos evaluar y escoger una metodología. Enfatiza la comunicación entre las partes interesadas. ADOO. Modelado de datos. Ciclo de vida estándar.	*Fases del ciclo de vida; técnicas de entrevista; JAD; DOO; prototipado; diseño de bases de datos; análisis de riesgos; gestión de proyectos; métricas de calidad del software; evaluación y adquisición de paquetes software; código de ética profesional.*
SI'97.8-Diseño físico e implementación con SGBD	Diseño e implementación de Sistemas de Información con un SGBD.	*Modelado de datos: herramientas y técnicas; paradigma estructurado y OO; modelos de bases de datos; CASE; repositorios; datawarehouse; IGU; cliente-servidor; conversiones; mantenimiento; formación de usuarios.*
SI'97.9-Diseño físico e implementación con entornos de programación	Diseño físico, programación, prueba e implantación de un sistema. Implementación OO y cliente-servidor utilizando entornos de desarrollo.	*Selección del entorno de programación cliente-servidor; construcción de software: estructurado, orientado a eventos y OO; pruebas; calidad del software; formación de usuarios; gestión de la configuración; mantenimiento; ingeniería inversa y reingeniería.*
SI'97.10-Gestión de proyectos y práctica	Cubre los factores necesarios de la gestión del desarrollo de sistemas. Cubre tanto los aspectos técnicos como los de comportamiento. Se centra en la gestión del desarrollo de sistemas de nivel empresarial.	*Gestión del ciclo de vida; integración de sistemas y bases de datos; gestión de redes y cliente-servidor; métricas para la gestión de proyectos y para la evaluación del rendimiento de sistemas; gestión de recursos humanos; análisis de coste-beneficio; técnicas de presentación; gestión de cambios.*

Tabla 4.15. Descripción de los cursos más relacionados con la Ingeniería del Software del IS'97

En el modelo curricular para graduados **MSIS 2000**, definido por la ACM y la AIS [Gorgone et al., 1999] presenta una línea profesional directamente relacionada con la Ingeniería del Software, **Analista y diseñador de sistemas**, que trataría temas de: *metodologías de diseño avanzadas (ADOO, RAD, Prototipado), Gestión de proyectos avanzada, Integración de sistemas* y *Consultoría de Sistemas de Información.*

La cronología de los diferentes currículos definidos en el campo de los Sistemas de Información comienza a comienzo de la década de los setenta con una propuesta curricular liderada por ACM. Los hechos más relevantes se presentan a continuación:

- Mayo de 1972: **ACM Graduate Professional Programs in Information Systems** [Ashenhurst, 1972].
- Diciembre de 1973: **ACM Undergraduate Programs in Information Systems** [Couger, 1973].
- Marzo de 1981: **ACM Educational Programs and Information Systems** [ACM, 1981].
- 1981 **DPMA Curriculum for Undergraduate Information Systems Education** [DPMA, 1981].
- En el año 1982, ACM elabora un nuevo informe para la enseñanza de los Sistemas de Información, el currículo **ACM-IS-82** [Nunamaker et al., 1982]. En él se hacen las siguientes distinciones entre el currículo en Ciencias de la Computación y el currículo para Sistemas de Información:
    - El currículo de Sistemas de Información enseña conceptos y procesos de Sistemas de Información, en dos contextos; conocimientos de organización y gestión, y conocimientos técnicos sobre Sistemas de Información. Por el contrario, las Ciencias de la Computación tienden a ser enseñadas en un entorno de matemáticas, algoritmos y tecnología.
    - En cuanto a conocimientos técnicos, el currículo de Sistemas de Información pone substancial énfasis en la capacidad para desarrollar la estructura de un Sistema de Información para una organización (institución/empresa) y para diseñar e implementar aplicaciones. Al titulado en Ciencias de la Computación se le suele hablar menos de análisis de requisitos de información y de consideraciones organizativas, pero adquiere mayores conocimientos en desarrollo de algoritmos, programación, software de sistemas y hardware.
- Durante 1990 se desarrolla el IS'90 propuesto por DPMA [Longenecker and Feinstein, 1991].
- En 1994 aparece el IS'94 de DPMA [Longenecker et al., 1994].
- Quince años después del currículo **ACM-IS-82**, en 1997 ACM vuelve a tratar los planes de estudio para los Sistemas de Información, cuando conjuntamente con la AITP (asociación norteamericana de profesionales de las tecnologías de la información, antes DPMA) y la AIS (asociación norteamericana de profesionales de los Sistemas de Información) propusieron un nuevo currículo para los Sistemas de Información, el IS'97 [Davis et al., 1997], de amplia repercusión.
- Actualmente se está trabajando una nueva actualización denominada IS'2000.
- Tanto el IS'97 como el IS'2000 son planes de estudios para estudiantes no graduados. Existe una propuesta de master especializado en Sistemas de Información, denominado MSIS2000 [Gorgone et al., 1999].

Cuadro 4.6. Cronología de los currículos en Sistemas de Información

4.4.3 Currículos centrados en la Ingeniería del Software

En su mayor parte, los currículos centrados en la Ciencia de la Informática están más orientados a la formación de científicos en computación que ingenieros, mientras que la realidad empresarial e industrial demanda profesionales que sean capaces de afrontar con éxito los proyectos que paulatinamente quedan inconclusos o con carencias significativas, suponiéndoles graves pérdidas económicas.

Para poder formar profesionales que afronten con garantías los problemas reales de las empresas se necesitan programas que hagan hincapié en la Ingeniería del Software, dado que los programas existentes centrados en las Ciencias de la Computación prestan poca atención a la Ingeniería del Software [Jalics and Golden, 1995], [Vaughn, 2000].

Existen diferentes propuestas curriculares que se centran en la Ingeniería del Software, ya sea en el contexto de los estudios de graduación o postgrado, cuya influencia está derivando en la definición de un cuerpo de conocimiento propio de la Ingeniería del Software que sentará la base para el establecimiento de una propuesta curricular de ámbito internacional que tenga a ésta como el objetivo central de la misma.

Se presentan a continuación las propuestas curriculares elaboradas por el **Instituto de Ingeniería del Software** (SEI – Software Engineering Institute) en la Universidad Carnegie-Mellon (CMU) en USA y por el **WGSEET** (Working Group on Software Engineering Education and Training).

4.4.3.1 Las propuestas del SEI-CMU

En 1985 la Universidad Carnegie Mellon presenta un currículo general en Informática [Shaw, 1985] donde la *Ingeniería del Software* aparece como una asignatura en el nivel intermedio, denominada "*Organización de programas*" (*"programming in the small", ampliación de TAD, POO, reusabilidad, especificaciones formales e informales...*) y en los cursos avanzados en forma de dos asignaturas: "*Ingeniería del Software*" (*"programming in the large", diseño avanzado y especificación, descomposición en módulos, CASE, prototipado, modelado...*) y "*Laboratorio de Ingeniería del Software*" (*desarrollo de proyectos en grupo, colaboración con la Industria*). En este currículo aparece el concepto de "*nociones recurrentes*" que después aparecerían como novedad en el currículo ACM/IEEE-CS91.

Con posterioridad el Instituto de Ingeniería del Software, SEI a partir de ahora, también en la Carnegie Mellon, realiza diferentes currículos específicos en Ingeniería del Software para estudios de graduación y de postgrado.

Programas de postgrado

El SEI establece primeramente estudios de postgrado, *masters*, centrados en Ingeniería del Software, que se detallan en diferentes informes y artículos entre 1989 y 1991

[Gibbs, 1989], [Ardis and Ford, 1989], [Ford, 1991a], siendo este último el de mayor difusión.

En el informe de 1991 se realiza un desarrollo muy completo y detallado de las áreas temáticas de Ingeniería del Software. El citado informe incluye un modelo de currículo, numerosa bibliografía sobre Ingeniería del Software y descripciones de revistas de investigación de la disciplina. Incluye también descripciones detalladas de los contenidos de la serie de audiovisuales (cintas de vídeo) que constituyen a su vez un ejemplo de una implementación del modelo de currículo. Se describen, además, los programas para postgraduados en Ingeniería del Software de 23 universidades de todo el mundo.

El material se organiza en cursos universitarios, e incorpora el concepto de "*unidad de conocimiento*", al igual que el ACM/IEEE-CS91. También como esta última propuesta, el currículo se organiza en una serie de materias fundamentales optativas avanzadas o complementarias (20-30% de un currículo), y proyectos prácticos de desarrollo (al menos un 30% del trabajo total del estudiante).

La secuencia que sugiere (aunque en esto no es restrictivo) es la de una aproximación docente, comenzando con una visión de proceso para colocar cada actividad individual en su contexto, para seguir con aspectos de gestión del software y actividades de control, finalizando con las actividades concretas de desarrollo y la visión producto.

En el currículo se incluyen materias tanto de Ingeniería del Software, como, de una forma más amplia, de Ingeniería de Sistemas. Las materias fundamentales se distribuyen en 21 unidades, sin recomendación temporal, describiendo para cada unidad sus contenidos, aspectos de la actividad que son más importantes y objetivos educativos de la unidad. A continuación se ofrece una relación de dichas unidades.

- El proceso de Ingeniería del Software
- Evolución del software
- Generación de software
- Mantenimiento de software
- Comunicación técnica
- Gestión de configuraciones software
- Conceptos de calidad del software
- Aseguramiento de la calidad del software
- Organización y gestión de proyectos de software
- Economía de proyectos software
- Conceptos de operación del software
- Análisis de requisitos
- Especificación
- Diseño de sistemas
- Diseño de software

- Implementación de software
- Pruebas del software
- Integración de sistemas
- Sistemas de tiempo real "empotrados"
- Interfaces hombre-máquina
- Asuntos de la profesión

Actualmente el master en Ingeniería del Software que se imparte en la Universidad Carnegie Mellon consta de tres elementos básicos [Garlan et al., 1997]:

1. **Un núcleo curricular:** Formado por los cursos sobre los que recaen los fundamentos de Ingeniería del Software, haciendo un énfasis especial en el análisis, diseño y gestión de grandes sistemas software. Los cursos que conforman este currículo son:
    a. **Modelos de sistemas software**
    b. **Métodos de desarrollo de software**
    c. **Gestión del desarrollo del software**
    d. **Análisis de elementos software**
    e. **Arquitecturas de sistemas software**
2. **Desarrollo de un proyecto:** Durante la duración del master, los alumnos planifican e implementan un proyecto software de tamaño significativo para un cliente externo. El trabajo se hace en equipo supervisado por profesores.
3. **Cursos de especialización:** De carácter optativo, que permiten que los estudiantes desarrollen una experiencia más profunda en una de las siguientes especialidades, entre las que se incluyen *sistemas de tiempo real, interfaces hombre-máquina* y *mejora del proceso software*.

Programas de graduación

En el SEI se llega a la conclusión de que la incesante demanda de ingenieros del software no se puede cubrir sólo con los estudiantes de postgrado, lo que hace necesaria la creación de programas de graduación en Ingeniería del Software. Estos programas quedan documentados en [Ford, 1990], [Ford, 1991b], [Ford, 1994].

El esquema de este currículo es:

1. **Matemáticas y Ciencia.** Con los objetivos de preparar a los alumnos para participar en una sociedad cada día más tecnológica, así como de ofrecerles los fundamentos necesarios para afrontar el resto de las asignaturas de la titulación. Se recomiendan dos asignaturas de matemáticas discretas, una de estadística y probabilidad, dos de cálculo, una de métodos numéricos, una de física, una de química y una de biología.

2. **Ciencia de Ingeniería y Diseño de Ingeniería.** Con los siguientes cursos:
    a. **Análisis.** Ofrece al alumno el conocimiento para realizar modelos y razonar sobre el proceso software. Algunos de los temas a tratar son: *desarrollo formal de algoritmos y programas (incluyendo la verificación formal de los mismos); técnicas de abstracción y modelado; sistemas formales y su aplicación a la Ingeniería del Software; métricas, análisis de algoritmos; análisis de rendimiento...*
    b. **Arquitecturas software.** Abordan la solución de problemas recurrentes a un alto nivel. Algunos temas pueden ser: *representación de datos, información y conocimiento; gestión de recursos; sistemas expertos; sistemas de tiempo real embebidos; sistemas concurrentes, paralelos o distribuidos...*
    c. **Hardware.** En la Ingeniería del Software no todo es software, debiendo un ingeniero del software comprender el hardware que está presente en los sistemas informáticos. Se tratan temas relacionados con *lenguajes ensambladores; arquitectura de computadores; sistemas digitales; redes...*
    d. **Proceso software.** Se encarga de transmitir el conjunto de herramientas, métodos y prácticas que se utilizan en la creación de software. Temas de este curso son: *análisis de requisitos; especificación y métodos formales; diseño; técnicas de implementación y lenguajes; verificación y validación; evolución del software; evaluación de productos y procesos software; gestión y organización de equipos de trabajo; temas éticos y profesionales.*
3. **Humanidades, Ciencias Sociales y Optativas.** Para completar la formación del alumno.

4.4.3.2 La propuesta del WGSEET

Cuando se presentaron los cuerpos de conocimiento sobre Ingeniería del Software, ya se comentó la iniciativa del WGSEET para la definición de un modelo de currículo para esta disciplina, que se recoge en [Bagert et al., 1999].

Esta propuesta curricular se centra en la generación de nuevos graduados que tengan en la Ingeniería del Software la base de conocimientos para afrontar una vida profesional condicionada por las necesidades de su entorno industrial y empresarial.

Arquitectura del currículo

En el diseño de este currículo intervienen una serie de elementos básicos, como se puede apreciar en la Figura 4.9.

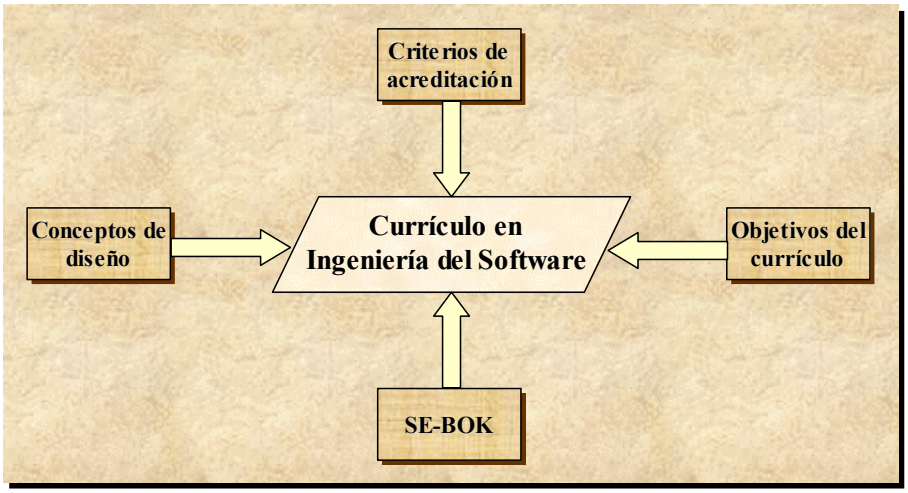

Figura 4.9. Arquitectura del currículo en Ingeniería del Software del WGSEET [Bagert et al., 1999]

La determinación de los objetivos del currículo se convierte en una actividad prioritaria en el diseño del mismo. Conlleva consideraciones sobre la misión de la institución que lo va a implantar y sobre la estrategia que se quiere seguir. En un nivel más fundamental, todos los afectados por la definición del currículo (*profesores, alumnos, empresarios...*) debieran estar representados en el momento de establecer estos objetivos.

El cuerpo de conocimientos ofrece la base de fundamentos para fijar el contenido del currículo, mientras que los criterios de acreditación introducen las bases para asegurar la calidad y la evaluación del mismo.

Por último, los conceptos de diseño construyen un marco filosófico para el desarrollo de currículos efectivos en Ingeniería del Software, así como para la definición del modelo de currículo del WGSEET.

Conceptos de diseño

La lista de conceptos que aparecen en esta sección constituye los principios básicos para el modelo de currículo en Ingeniería del Software propuesto por el WGSEET.

1. Soporta el desarrollo de varios programas de estudios, con la característica común de centrarse en la Ingeniería del Software (*Ingeniería del Software, Ciencias de la Computación, Sistemas de Información...*).
2. Introduce a los alumnos, desde el comienzo de sus estudios, en la naturaleza, ámbito e importancia de la Ingeniería del Software.
3. El modelo incorpora dos niveles de educación en Ingeniería del Software:
    a. La denominada Ingeniería del Software a pequeña escala (*Software Engineering in the Small*)

i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados de forma individual o en pequeños grupos.

ii. El desarrollo del software con estas características se encuentra concentrado en los primeros cursos.

b. La denominada Ingeniería del Software a gran escala (*Software Engineering in the Large*)

i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados en equipo.

ii. El desarrollo del software con estas características se encuentra localizado en cursos avanzados o en pequeños proyectos realizados para empresas.

4. El modelo establece un balance entre *producto* y *proceso*.

   a. Las actividades relacionadas con el producto incluyen métodos, técnicas y propiedades usados en la construcción de los elementos software asociados en un producto software (*planes de desarrollo, planes de aseguramientos de la calidad, especificación de requisitos, especificación de diseño, código, documentación...*).

   b. Las actividades relacionadas con el proceso incluyen los estándares, procedimientos, guías, métricas y técnicas de análisis y soporte que ofrecen el marco adecuado para llevar a cabo las actividades de construcción del producto de forma efectiva y eficiente.

5. El modelo ofrece una guía para el desarrollo de programas de graduación que pueden ser acreditados por organizaciones externas.

6. Se incluyen asuntos éticos, sociales y profesionales que estén directamente relacionados con la práctica profesional de la Ingeniería del Software.

7. El modelo enfatiza el concepto de trabajo en grupo.

8. Ofrece especialización en dominios de aplicación concretos (*sistemas empotrados, bases de datos y Sistemas de Información, sistemas inteligentes, redes...*).

9. Asegura la práctica de la Ingeniería del Software:

   a. Prácticas en laboratorios planificadas.

   b. Proyectos de Ingeniería del Software a realizar en laboratorio.

   c. Educación cooperativa y prácticas en empresas.

Contenidos del currículo

Aunque la organización y puesta en marcha de varios currículos centrados en la Ingeniería del Software puede diferir de unos a otros, todos ellos deben ofrecer asignaturas que incluyan conocimientos sobre *Matemáticas, Fundamentos de Ciencias de la Computación, Ingeniería del Software, Educación General, Ciencias Naturales* y *Conocimientos de Dominios Específicos*.

En la siguiente lista se describe lo que, como mínimo, el WGSEET considera que se debe incluir en cada área:

1. **Fundamentos de Ciencias de la Computación**
    a. Programación, estructuras de datos y algoritmos.
    b. Conceptos de lenguajes de programación.
    c. Organización de computadores.
    d. Sistemas software (*gestión de procesos y recursos, computación concurrente y distribuida, redes*).
2. **Matemáticas**
    a. Matemática discreta.
    b. Estadística y probabilidad.
3. **Ciencias naturales**
4. **Educación general**
    a. Comunicación (oral y escrita).
    b. Ciencias sociales y humanidades.
5. **Ingeniería del Software**
    a. Ingeniería de requisitos.
    b. Diseño del software.
    c. Calidad del software.
    d. Arquitecturas software.
    e. Construcción del software.
    f. Evolución del software.
    g. Métodos formales.
    h. Interfaces hombre-máquina.
    i. Organización, planificación de proyectos.
    j. Proceso software.
    k. Ética y profesionalidad en la Ingeniería del Software.

Módulos propios de la Ingeniería del Software

Los módulos que se presentan en la Tabla 4.16, pueden hacerse corresponder cada uno de ellos con una sola asignatura, o combinarse varios de ellos en una asignatura.

Módulo	Título	Descripción
IS1	*Introducción a la Ciencia de la Informática para ingenieros del software 1*	Introducción a la programación, normalmente en el paradigma procedimental. Las características básicas de la Ingeniería del Software se integran en este curso.
IS2	*Introducción a la Ciencia de la Informática para ingenieros del software 2*	Introducción a las estructuras de datos y al paradigma objetual. Se continúa con la introducción de conceptos de Ingeniería del Software.
IS3	*Introducción a la Ingeniería del Software*	Descripción general de la Ingeniería del Software como disciplina; introduce los principios fundamentales y las metodologías.
IS4	*Ética y profesionalismo*	Cubre material sobre aspectos históricos, sociales y económicos de la Ingeniería del Software. Incluye el estudio de las responsabilidades y riesgos profesionales, así como sobre la propiedad intelectual.
IS5	*Requisitos del software*	Introduce los conceptos básicos y principios de la ingeniería de requisitos: sus herramientas, técnicas y métodos para el modelado del software.
IS6	*Diseño del software*	Métodos y técnicas utilizados en la fase de diseño. Énfasis en el diseño orientado a objetos.
IS7	*Calidad del software*	Aseguramiento de la calidad y gestión de la configuración.
IS8	*Construcción y evolución del software*	Examina problemas, métodos y técnicas asociadas con la construcción del software, dado un diseño de alto nivel y teniendo en cuenta el mantenimiento del mismo.
IS9	*Proyecto*	Permite a los estudiantes poner en práctica los conocimientos adquiridos en el resto de módulos, al entrar a formar parte de un equipo de desarrollo que se encarga de la realización de un proyecto.

Tabla 4.16. Módulos relacionados con la Ingeniería del Software en el currículo WGSEET

Para una descripción más detallada de cada uno de estos módulos, incluyendo un índice de primer nivel de los contenidos de cada uno de ellos, se recomienda la consulta de [Bagert et al., 1999].

Influencias de otras propuestas curriculares

La propuesta curricular del WGSEET recibe influencias de otras propuestas anteriores, que también coinciden en el objetivo de establecer un currículo independiente para la disciplina de Ingeniería del Software. De todas ellas se van presentar las dos más relevantes.

Thomas B. Hilburn propone un modelo conceptual de currículo en Ingeniería del Software [Hilburn, 1997] que se basa en tres pilares fundamentales:

- ***Las personas:*** Un currículo debe soportar las actividades que formen al alumno para trabajar y comunicarse con sus semejantes de una forma efectiva. Las capacidades a considerar son: *educación general, capacidad de comunicación, trabajo en grupo* así como *ética y profesionalismo.*
- ***Proceso:*** Los estudiantes deben darse cuenta de la necesidad de utilizar un proceso definido para desarrollar productos software, distinguiendo el proceso para la creación de software de carácter individual, el proceso para trabajar en equipo y el proceso que involucra a los estándares seguidos por una organización.
- ***Tecnología:*** Los alumnos deben adquirir los conocimientos tecnológicos y científicos necesarios para el desarrollo de software de calidad, incluyendo conocimientos de *Matemáticas y Ciencia, Ciencias de la Computación* y *Desarrollo de Software*.

A. J. Cowling propone un modelo de currículo en Ingeniería del Software multi-dimensional [Cowling, 1998], que establece las siguientes dimensiones:

- **Los diferentes niveles de abstracciones que definen los componentes hardware y software**. Establece que el ámbito de la Ingeniería del Software está en la construcción de sistemas en los que su parte principal está constituida por componentes software. Permite distinguir a los ingenieros del software de aquéllos que trabajan en la parte de arquitecturas de computadores.
- **El balance de los contenidos en Informática con respecto a otras ramas de la Ciencia y la Ingeniería**. Permite distinguir entre la Ingeniería del Software y otras disciplinas.
- **El balance entre la teoría, el modelado y su aplicación práctica**. La base de esta tercera dimensión es la observación de que la Ingeniería del Software no es la mera construcción de sistemas software, sino que la construcción de éstos se hace acorde a un método de Ingeniería.
- **El balance entre la parte técnica y la parte no técnica**. Las tres primeras dimensiones se centran en la parte técnica, pero una característica importante de la Ingeniería es que consiste en algo más que un conjunto de aspectos técnicos. Se necesitan contemplar aspectos económicos, sociales y sicológicos.

4.5 Referencias

[Abran et al., 1999] Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.5. ACM/IEEE-CS, October, 1999.

[Abran et al., 2000] Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.6. ACM/IEEE-CS, February, 2000. Available on line at http://www.swebok.org [Última vez visitado 21-3-2000].

[ACM, 1968] ACM Curriculum Committee on Computer Science. "*Curriculum '68: Recommendations for Academic Programs in Computer Science*". Communications of the ACM, 11(3):151-197. March, 1968.

[ACM, 1981] ACM Committee on Computer Curricula of ACM Education Board. "*ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities*". ACM, New York, 1981.

[ACM/IEEE-CS, 1998] ACM/IEEE-CS. "*Accreditation Criteria for Software Engineering*". http://www.acm.org/serving/se/Accred.htm. [Última vez visitado 28/12/1999]. September, 1998.

[ACM/IEEE-CS, 1999a] ACM/IEEE-CS. "*Software Engineering Education Project*". http://www.acm.org/serving/se/SWEE.htm. [Última vez visitado 28/12/1999]. 1999.

[ACM/IEEE-CS, 1999b] ACM/IEEE-CS. "*1999 Plan for the Software Engineering Education Project (SWEEP)*". Draft 0.5. http://www.acm.org/serving/se/sweep.htm. [Última vez visitado 28/12/1999]. April, 1999.

[ACM/IEEE-CS, 1999c] ACM/IEEE-CS. "*Software Engineering Code of Ethics and Professional Practice*". Version 5.2. http://computer.org/tab/sweec/code.htm. [Última vez visitado 28/12/1999]. 1999.

[Ada95-Web] "*Ada 95 Reference Manual: Language and Standard Library*". http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-toc.html [Última vez visitado 8/1/2000].

[Adams, 1996] Adams, Joel C. "*Object-Centered Design. A Five-Phase Introduction to Object-Oriented Programming in CS1-2*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 78-82. ACM. 1996.

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[Alhir, 1998] Alhir, Sinan Si. "*The Object-Oriented Paradigm*". http://home.earthlink.net/~salhir/theobjectorientedparadigm.html. [Última vez visitado 23/12/1999]. October, 1998.

[Apple, 1989] Apple Computer Inc. "*Macintosh Programmer's Workshop Pascal 3.0 Reference*". Apple Computer, 1989.

[Ardis and Ford, 1989] Ardis, Mark and Ford, Gary. "*1989 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-89-TR-21 (ESD-TR-89-

29), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). June, 1989.

[Arnold and Gosling, 1997] Arnold, Ken and Gosling, James. "*The Java Programming Language*". 2nd edition. Addison-Wesley, 1997.

[Ashenhurst, 1972] Ashenhurst, R. L. (Editor). "*Curriculum Recommendations for Graduate Professional Programs in Information Systems*". ACM, 1972.

[Ashworth and Goodland, 1990] Ashworth, C. and Goodland, M. "*SSADM: A Practical Approach*". McGraw-Hill, 1990.

[Atkinson and Buneman, 1987] Atkinson, M. and Buneman, P. "*Types and Persistence in Database Programming Languages*". ACM Computing Surveys, 19(2). 1987.

[Atkinson et al., 1989] Atkinson, Malcom, Bancilhon, François, DeWitt, David, Dittrich, Klaus, Maier, David, Zdonik, Stanley. "*The Object-Oriented Database System Manifesto*". In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto (Japan). 1989. Also in Deductive and Object-Oriented Databases, Elsevere Science Publishers, Amsterdam, Netherlands, 1990.

[Austing et al., 1979] Austing, Richard, Barnes, Bruce, Bonnette, Della, Engel, Gerald and Stokes, Gordon. "*Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*". Communications of the ACM, 22(3):147-166. March, 1979.

[Bagert et al., 1999] Bagert, Donald J., Hilburn, Thomas B., Hislop, Greg, Lutz, Michael, McCracken, Michael and Mengel, Susan. "*Guidelines for Software Engineering Education. Version 1.0*". Working Group on Software Engineering Education and Training (WGSEET). August, 1999.

[Bailin, 1989] Bailin, Sidney C. "*An Object-Oriented Requirements Specification Method*". Communications of the ACM, 32(5):608-623. Mayo, 1989.

[Barr and Feigenbaum, 1981] Barr, A. and Feigenbaum, E. "*The Handbook of Artificial Intelligence*". Vol. 1. William Kaufmann, 1981.

[Basili, 1991] Basili, V. R. "*The Future Engineering of Software: A Management Perspective*". IEEE Computer, 24(9):90-96. September, 1991.

[Bauer, 1972] Bauer, F. L. "*Software Engineering*". Information Processing 71. Amsterdam: North Holland, 1972.

[BCS, 1989] The British Computer Society and The Institution of Electrical Engineering. "*A Report on Undergraduate Curricula for Software Engineering Curricula*". June, 1989.

[Bellin, 1999] Bellin, David. "*Pedagogical Pattern #4. Brainstorming Pattern*". Version 2.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp4.htm. [Última vez visitado, 20/8/1999]. July, 1999.

[Bergin, 1998a] Bergin, Joseph. "*Pedagogical Patterns*". http://csis.pace.edu/~bergin/PedPat1.2.html. [Última vez visitado, 3/8/1999]. October, 1998.

[Bergin, 1998b] Bergin, Joseph. "*Six Pedagogical Patterns*". http://csis.pace.edu/~bergin/fivepedpat.html. [Última vez visitado, 3/8/1999]. October, 1998.

[Bergin, 1998c] Bergin, Joseph. "*Pedagogical Pattern #32. Spiral Pattern*". Versión 1.2. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp32.htm. [Última vez visitado, 20/8/1999]. October, 1998.

[Bertino and Martino, 1993] Bertino, E. and Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

[Bertolino, 1999] Bertolino, A. "*KA Description of Software Testing V. 0.5*". In [Abran et al., 1999], 1999.

[Bézivin et al., 1992] Bézivin, Jean, Roux, Olivier and Royer, Jean-Claude. "*Teaching Object-Oriented Programming or Using the Object Model to Teach Software Engineering*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 269-275. ACM. 1992.

[Birtwistle et al., 1973] Birtwistle, Graham M., Dahl, Ole-Johan, Myhrhaug, Bjorn and Nygaard, Kristen. "*Simula Begin*". Studentliteratur, 1973.

[Blaha and Premerlani, 1998] Blaha, Michael and Premerlani, William. "*Object-Oriented Modeling and Design for Database Applications*". Prentice Hall, 1998.

[Blaha et al., 1988] Blaha, Michael, Premerlani, William and Rumbaugh, J. "*Relational Database Design Using an Object-Oriented Methodology*". Communications of the ACM, 31(4):414-427. April, 1988.

[Blair et al., 1991] Blair, G., Gallagher, J., Hutchinson, D. and Shepard, D. "*Object-Oriented Languages, Systems and Applications*". Halsted Press, 1991.

[Blum, 1992] Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, 1992.

[Bobrow and Stefik, 1982] Bobrow, Daniel G. and Stefik, Mark J. "*LOOPS: an Object-Oriented Programming System for Interlisp*". Xerox PARC, 1982.

[Boehm, 1976] Boehm, B. W. "*Software Engineering*". IEEE Transactions on Computers. C-25(12). December, 1976.

[Bollinguer, 1999] Bollinguer, Terry. "*Software Construction (Versión 0.5)*". In [Abran et al., 1999], 1999.

[Booch, 1991] Booch, Grady. "*Object Oriented Design with Applications*". The Benjamin/Cummings Publishing Company, 1991.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Booch and Rumbaugh, 1995] Booch, Grady and Rumbaugh, James. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

[Booch et al., 1996a] Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

[Booch et al., 1996b] Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.91 Addendum UML update. Rational Software Corporation, September 1996.

[Booch et al., 1997a] Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997.

[Booch et al., 1997b] Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0.1. Rational Software Corporation, 19 March 1997.

[Booch et al., 1999] Booch, Grady, Rumbaugh, James and Jacobson, Ivar. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

[Bourque et al., 1998] Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W., Tripp, Leonard, Shyne, Karen, Pflug, Bryan, Maya, Marcela and Tremblay, Guy. "*Guide to the Software Engineering Body of Knowledge. A Straw Man Version*". ACM/IEEE-CS, September, 1998.

[Bourque et al., 1999a] Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W. and Tripp, Leonard. "*The Guide to the Software Engineering Body of Knowledge*". IEEE Software, 16(6):35-44. November-December, 1999.

[Bourque et al., 1999b] Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W., Tripp, Leonard and Frailey, Dennis. "*Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge*". Technical Report. January, 1999.

[Bowyer, 1996] Bowyer, Kevin W. "*Ethics and Computing: Living Responsibly in a Computerized World*". IEEE Computer Society Press, 1996.

[Budd, 1991] Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Buxton and Randell, 1970] Buxton, J. N. and Randell, B. (Editors). "*Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 October, 1969*". Brussels: Scientific Affairs Division, NATO. April, 1970.

[Buxton et al., 1976] Buxton, J. M., Naur, P. and Randell, B. (Editors) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[Cameron, 1989] Cameron, J. "*JSP & JSD: The Jackson Approach to Software Development*". 2nd edition. IEEE Computer Society Press, 1989.

[Camps, 1999] Camps Paré, Rafael. "*¿Qué Informática Se Enseña en la Universidad? (Primera Parte)*". Novática. Nº 141: 48-51. Septiembre/Octubre, 1999.

[Cannon, 1980] Cannon, H. I. "*Flavors*". Technical Report, MIT Artificial Intelligence Laboratory, Cambridge (Mass.), 1980.

[Carrington, 1999] Carrington, David. "*SWEBOK Knowledge Area Description for Software Engineering Infraestructure (version 0.5)*". In [Abran et al., 1999], 1999.

[Castellanos et al., 1991] Castellanos, M. G., Saltor, F. and García-Solaco, M. "*The Development of Semantic Concepts in BLOOM Model Using an Object Metamodel*". Technical Report LSI-91-22. Dept. de Llenguatges i Sistemes Informátics. Universidad Politécnica de Cataluña, 1991.

[Coleman et al., 1994] Coleman, D., Arnold, P., Bodoff, S., Dolin, C., Hayes, F. and Jeremaes, P. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

[Cook and Daniels, 1994] Cook, S. and Daniels, J. "*Designing Object Systems: Object Oriented Modelling with Syntropy*". Prentice-Hall, 1994.

[Cook et al., 1997] Cook, Steve, Selic, Bran, Gangopadhyay, Dipayan, Gheorge, Serban, Gullekson, Garth, Hogg, John, McGee, Jim, Meier, Mike, Mitra, Subrata, Saaltink,

Mark, Warmer Jos and Wills Alan. "*OMG OA&D RFP OMG OA&D RFP Response*". Document Version 1.0. IBM Corporation and ObjecTime Limited. 10 January 1997.

[Couger, 1973] Couger, J. (Editor) "*Curriculum Recommendations for Undergraduate Programs in Information Systems*". Communications of the ACM, 16(12): 727-749. December, 1973.

[Cowling, 1998] Cowling, A. J. "*A Multi-Dimensional Model of the Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education and Training – CSEE&T'98. (February 22-25, 1998, Atlanta, GA, USA). Pages 44-55. IEEE Computer Society. 1998.

[Cox, 1984] Cox, Brad J. "*Message/Object Programming: An Evolutionary Change in Programming Technology*". IEEE Software, 1(1):50-69. January, 1984.

[Cox and Novobilski, 1990] Cox, Brad J. and Novobilski, Andrew J. "*Object-Oriented Programming: An Evolutionary Approach*". 2nd edition. Addison-Wesley, 1990.

[Champeaux et al., 1993] Champeaux, Dennis, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison Wesley. 1993.

[Chang et al., 1999] Chang, Carl K., Engel, Gerald, King, Willis, Roberts, Eric, Shackelford, Russ, Sloan, Robert H. and Srimani, Pradip K. "*Curricula 2001: Bringing the Future to the Classroom*". Computer, 32(9):85-88. September, 1999.

[Chen, 1976] Chen, Peter. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March, 1976.

[Dahl and Hoare, 1972] Dahl, Ole-Johan and Hoare, C. A. R. "*Hierarchical Program Structures*". In Dahl, Dijkstra, Hoare, *Structured Programming*, Academic Press, pages 175-220. 1972.

[Dahl and Nygaard, 1966] Dahl, Ole-Johan and Nygaard, Kristen. "*SIMULA — An Algol-Based Simulation Language*". Communication of the ACM, 9(9):671-678. September, 1966.

[Dahl et al., 1970] Dahl, Ole-Johan, Myrhaug, Bjorn and Nygaard, Kristen. "*(Simula 67) Common Base Language*". Norsk Regnesentral (Norwegian Computing Center), Publication N. S-22, Oslo. October, 1970.

[Davis, 1993] Davis, Alan M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

[Davis et al., 1997] Davis, Gordon B., Gorgone, John T., Couger, J. Daniel, Feinstein, David L. and Longenecker, Jr. Herbert E. (Editors) "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS and AITP, 1997.

[Decker and Hirshfield, 1994] Decker, Rick and Hirshfield, S. "*The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS1*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 51-55. ACM. 1994.

[DeMarco, 1979] DeMarco, Tom "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

[Denning, 1992] Denning, Peter J. "*Educating a New Engineer*". Communications of the ACM, 35(12). December, 1992.

[Denning et al., 1988] Denning, Peter J., Comer, Douglas E., Gries, David, Mulder, Michael C., Tucker, Allen B., Turner, A. Joe and Young, Paul R. "*Report of the ACM Task Force on the Core of Computer Science*". ACM Press, 1988.

[Denning et al., 1989] Denning, Peter J., Comer, Douglas E., Gries, David, Mulder, Michael C., Tucker, Allen B., Turner, A. Joe and Young, Paul R. "*Computing as a Discipline*". Communications of the ACM, 32(1):9-23. January, 1989.

[Dijkstra, 1968] Dijkstra, E. "*The Structure of 'THE' Multiprogramming System*". Communications of the ACM, 11(5). May, 1968.

[Diller, 1990] Diller, A. "*Z: An Introduction to Formal Methods*". John Wiley & Sons, 1990.

[Dodani, 1999] Dodani, Mahesh. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January, 1999.

[Dolado, 1999] Dolado, Javier. "*El Código de Ética y Práctica Profesional de la Ingeniería del Software de la ACM/IEEE Computer Society*". Novática. Nº 140. Julio-Agosto, 1999.

[Dorling, 1993] Dorling, Alec. "*Software Process Improvement and Capability Determination*". Software Quality Journal, 12(4): 209-224. December, 1993.

[DPMA, 1981] Data Processing Management Association. "*DPMA Model Curriculum, 1981*". Published by DPMA, Chicago, 1981.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Duncan, 1996] Duncan, William R. "*A Guide to the Project Management Body of Knowledge*". PMI Standards Committee. Project Management Institute, Four Campus Boulevart, Newtown Square, PA 19073-3299, USA. 1996.

[D'Souza, 1996] D'Souza, Desmond F. "*Objects: Education vs. Training*". ICON Computing Inc. http://www.iconcomp.com/papers/education-vs-training/EducationvsTraining.frm.html. [Última vez visitado, 24/5/1999]. 1996.

[D'Souza and Wills, 1999] D'Souza, Desmond F. and Wills, Alan Cameron. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

[Education Activities Board, 1983] Education Activities Board. "*The 1983 Model Program in Computer Science and Engineering*". Technical Report 932. IEEE Computer Society. December, 1983.

[Ehrig and Mahr, 1985] Ehrig, H. and Mahr, B. "*Fundamentals of Algebraic Specification 1*". Springer-Verlag, EATCS Nº6, 1985.

[Ellis and Stroustrup, 1990] Ellis, Margaret and Stroustrup, Bjarne. "*The Annotated C++ Reference Manual*". Addison Wesley, 1990.

[Fairley, 1985] Fairley, R. "*Software Engineering Concepts*". McGraw-Hill, 1985.

[Firesmith et al., 1998] Firesmith, Donald, Henderson-Sellers, Brian and Graham, Ian. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

[Ford, 1990] Ford, Gary. "*1990 SEI Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-90-TR-3 (ESD-TR-90-204), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March, 1990.

[Ford, 1991a] Ford, Gary. "*1991 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-91-TR-2 (ESD-TR-91-2), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April, 1991.

[Ford, 1991b] Ford, Gary. "*The SEI Undergraduate Curriculum in Software Engineering*". In Proceedings of the twenty-second SIGCSE technical symposium on Computer Science Education – SIGCSE'91. (March 7-8, 1991, San Antonio, Texas, USA). Pages 375-385. ACM. 1991.

[Ford, 1994] Ford, Gary. "*A Progress Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-94-TR-11 (ESC-TR-94-011), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). May, 1994.

[Frakes et al., 1991] Frakes, William B., Fox, Christopher, Nejmeh and Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

[Gane and Sarson, 1977] Gane, C. and Sarson, T. "*Structured Systems Analysis and Design*". Improved Systems Technologies, Inc., 1977.

[Gane and Sarson, 1979] Gane, C. and Sarson, T. "*Structured Systems Analysis: Tools and Techniques*". Prentice-Hall, 1979.

[García y Pardo, 1998] García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Garlan et al., 1997] Garlan, David, Gluch, David P. and Tomayko, James E. "*Agents of Change: Educating Software Engineering Leaders*". IEEE Computer, 30(11):59-65 November, 1997.

[Gibbs, 1989] Gibbs, Norman E. "*The SEI Education Program: The Challenge of Teaching Future Software Engineers*". Communications of the ACM, 32(5):594-605. May, 1989.

[Gibbs and Tucker, 1986] Gibbs, N. E. and Tucker, A. B. "*Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 29(3):202-210. March, 1986.

[Gómez et al., 1998] Gómez, A., Juristo, N., Montes, C. y Pazos, J. "*Ingeniería del Conocimiento*". Madrid. Ceura, 1998.

[Goguen and Meseguer, 1988] Goguen, J. A. and Meseguer, J. "*Order-Sorted Álgebra I*". Technical Report, SRI International, Stanford University, 1988.

[Goguen et al., 1992] Goguen, J. A., Winkler, T., Meseguer, J., Futatsugui, K. and Jouannaud, J. P. "*Introducing OBJ*". SRI-CSL Report. Draft of January, 1992.

[Goldberg and Kay, 1976] Goldberg, Adele and Kay, Alan. "*Smalltalk-72 Instruction Manual*". Technical Report SSL-76-6, Xerox Palo Alto Research Center. March, 1976.

[Goldberg and Robson, 1983] Goldberg, Adele and Robson, David. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Goldberg, 1985] Goldberg, Adele. "*Smalltalk-80: The Interactive Programming Environment*". Addison-Wesley, 1985.

[Goldberg, 1986] Goldberg, R. "*Software Engineering: An Emerging Discipline*". IBM Systems Journal. 25(3/4), 1986.

[Gorgone et al., 1999] Gorgone, John T., Gray, Paul, Feinstein, David L., Kasper, George M., Luftman, Jerry N., Stohr, Edward A., Valacich, Joseph and Wigand, Rolf T.

"*MSIS 2000 Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems*". ACM and AIS. November, 1999.

[Gotterbarn, 1999] Gotterbarn, D. "*How the New Software Engineering Code of Ethics Affects You*". IEEE Software, 16(6):58-64. November/December, 1999.

[Gotterbarn et al., 1997a] Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics*". Communications of the ACM, 40(11):110-118. November, 1997.

[Gotterbarn et al., 1997b] Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics, Version 3.0*". IEEE Computer, 30(11):88-92. November, 1997.

[Gotterbarn et al., 1999a] Gotterbarn, D., Miller, K. and Rogerson, S. "*Computer Society and ACM Approve Software Engineering Code of Ethics*". IEEE Computer, 32(10):84-88. October, 1999.

[Gotterbarn et al., 1999b] Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics Is Approved*". Communications of the ACM, 42(10):102-107. October, 1999.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd edition. Addison-Wesley, 1994.

[Graham et al., 1997] Graham, Ian, Henderson-Sellers, Brian and Younessi, Houman. "*The Open Process Specification*". Addison Wesley (Open Series), 1997.

[Guttag, 1980] Guttag, J. "*Abstract Data Types and the Development of Data Structures*". In *Programming Language Design*. Computer Society Press, 1980.

[Hadjerrouit, 1999] Hadjerrouit, Said. "*A Constructivist Approach to Object-Oriented Design and Programming*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 171-174. ACM. 1999.

[Harbison, 1992] Harbison, Samuel P. "*Modula-3*". Prentice Hall, 1992.

[Hatley and Pirbhai, 1987] Hatley D. J. and Pirbhai, A. "*Strategies for Real-Time System Specification*". Dorset House, 1987.

[Henderson-Sellers and Edwards, 1994a] Henderson-Sellers, B. and Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

[Henderson-Sellers and Edwards, 1994b] Henderson-Sellers, B. and Edwards, J. M. "*MOSES: A Second Generation Object-Oriented Methodology*". Object Magazine, pp. 68-73. June, 1994.

[Henderson-Sellers et al., 1998] Henderson-Sellers, Brian, Simons, Anthony, Younessi, Houman. "*The Open Toolbox of Techniques*". Open Series. Addison Wesley, 1998.

[Hilburn, 1997] Hilburn, Thomas B. "*Software Engineering Education: A Modest Proposal*". IEEE Software, 14(6):44-48. November/December, 1997.

[Hilburn et al., 1998] Hilburn, Thomas B., Bagert, Donald J., Mengel, Susan and Oexmann, Dale. "*Software Engineering Across Computing Curricula*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 117-121. ACM. 1998.

[Hilburn et al., 1999] Hilburn, Thomas B., Hirmanpour, Iraj, Khajenoori, Soheil, Turner, Richard and Qasem, Abir. "*A Software Engineering Body of Knowledge Version 1.0*". Technical Report CMU/SEI-99-TR-004 (ESC-TR-99-004), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April, 1999.

[Hirmanpour et al., 1995] Hirmanpour, Iraj, Hilburn, Thomas B. and Kornecki, Andrew. "*A Domain Centered Curriculum. An Alternative Approach to Computing Education*". In Proceedings of the 26th SISCSE technical symposium on Computer Science Education, SIGCSE '95. (March 2-4, 1995, Nashville, TN, USA). ACM. 1995.

[Hoare, 1985] Hoare, C. A. R. "*Communicating Sequential Processes*". Prentice-Hall, 1985.

[Holland et al., 1997] Holland, Simon, Griffiths, Robert and Woodman, Mark. "*Avoiding Object Misconceptions*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 131-134. 1997.

[Horan, 1995] Horan, Peter "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[Hullot, 1984] Hullot, Jean-Marie. "*Ceyx, Version 15: I — une Initiation*". Rapport Technique no. 44, INRIA, Rocquencourt, 1984.

[Humphrey, 1989] Humphrey, W. S. "*Managing the Software Process*". Addison-Wesley, 1989.

[Humphrey, 1993] Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science,* Van Nostrand Reinhold, p. 1218,1993.

[IEEE, 1983] IEEE. "*Standard Glossary of Software Engineering Terminology*". ANSI/IEEE Std. 729-1983. IEEE, 1983.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[Ingalls, 1978] Ingalls, Daniel H. "*The Smalltalk-76 Programming System: Design and Implementation*". In Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages. ACM. January, 1978.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[ISO/IEC, 1998] ISO/IEC. "*Programming Languages – C++*". Technical ISO/IEC 14882. September, 1998.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Jacobson et al., 1999] Jacobson, I., Booch, G. and Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

[Jackson, 1983] Jackson, M. A. "*System Development*". Prentice-Hall, 1983.

[Jackson et al., 1997] Jackson, Ursula, Manaris, Bill and McCauley, Renée. "*Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum.*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education, SIGCSE '97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 360-364. ACM. 1997.

[Jalics and Golden, 1995] Jalics, P. and Golden, D. "*A Profile of Undergraduate Computer Science Curricula*". Computer Science Education, 6(2):179-192. November, 1995.

[Jalloul, 1999] Jalloul, Ghinwa. "*Pedagogical Pattern #48. Academic to Industrial Project Link (LINK) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp48.htm. [Última vez visitado, 20/8/1999]. July, 1999.

[Jaworski, 1998] Jaworski, Jaime. "*Java 1.2 Unleashed*". Sams, 1998.

[Jones, 1987] Jones, A. K. "*The Object Model: A Conceptual Tool for Structuring Software*". In Gerald E. Peterson, editor, *TUTORIAL: Object-Oriented Computing*, volume 2: Implementations. IEEE Computer Society Press, 1987.

[Jones, 1990] Jones, C. B. "*Systematic Software Construction Using VDM*". Prentice-Hall, 1990.

[Joyner, 1996] Joyner, Ian. "*C++?? A Critique of C++ and Programming and Language Trends of the 1990s*". 3rd edition http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/cppcv3.pdf. [Última vez visitado 7/1/2000]. 1996.

[Joyner, 1999] Joyner, Ian. "*Objects Unencapsulated. Eiffel, Java and C++??*". Object and Component Technology Series, Prentice-Hall, 1999.

[Knight et al., 1994] Knight, John C., Prey, Jane C. and Wulf, Wm. A. "*Undergraduate Computer Science Education: A New Curriculum Philosophy & Overview*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 155-159. ACM. 1994.

[Kobryn, 1999] Kobryn, Cris. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October, 1999.

[Koffman et al., 1984] Koffman, E., Miller, P. and Wardle, C. "*Recommended Curriculum for CS1: 1984*". Communications of the ACM, 27(10):998-1001. October, 1984.

[Koffman et al., 1985] Koffman, E., Miller, P. and Wardle, C. "*Recommended Curriculum for CS2: 1984*". Communications of the ACM, 28(8):815-818. August, 1985.

[Konrad et al., 1995] Konrad, Michael D., Paulk, Mark C., and Graydon, Allan W. "*An Overview of SPICE's Model for Process Management*". In Proceedings of the Fifth International Conference on Software Quality, Austin, TX, 23-26 October 1995.

[Krasner and Pope, 1988] Krasner, G. E. and Pope, S. T. "*A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*". Journal of Object-Oriented Programming (JOOP), SIGS Publications, 1(3):26-49. August-September, 1988.

[Lalonde and Pugh, 1990] Lalonde, Wilf R. and Pugh, John R. "*Inside Smalltalk*". Vol. 1. Prentice-Hall, 1990.

[Lalonde and Pugh, 1991] Lalonde, Wilf R. and Pugh, John R. "*Inside Smalltalk*". Vol. 2. Prentice-Hall, 1991.

[Larman, 1998] Larman, Craig. "*Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*". Prentice Hall, 1998.

[Layman, 1994] Layman, B. "*ISO-9000 Standards and Existing Quality Models: How They Relate*". American Programmer Review, pp. 9-15. February, 1994.

[Le Moigne, 1973] Le Moigne, J. L. "*Les Systemes D'Information dan les Organizations*". Presses Universitaires de France, 1973.

[Lebsanft and Synspace, 1994] Lebsanft, E. and Synspace, A. G. "*BOOTSTRAP: Experiences with Europe's Software Process Assesment & Improvement Method*". In Proceedings of the 1st World Congress for Software Quality, 1994.

[Levine et al., 1991] Levine, Linda, Pesante, Linda H. and Dunkle, Susan B. "*Technical Writing for Software Engineers*". Curriculum Module, SEI-CM-23, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). November, 1991.

[Levy, 1984] Levy, H. "*Capability-Based Computer Systems*". Digital Press, 1984.

[Liskov and Zilles, 1977] Liskov, B. and Zilles, S. "*An Introduction to Formal Specifications of Data Abstractions*". Current Trends in Programming Methodology: Software Specification and Design. Vol. 1. Prentice-Hall, 1977.

[Longenecker and Feinstein, 1991] Longenecker, Jr. Herbert E. and Feinstein, David L. (Editors) "*IS'90 The DPMA Model Curriculum for a Four Year Undergraduate Degree for the 1990s*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1991.

[Longenecker et al., 1994] Longenecker, Jr. Herbert E., Fournier, Robert, Reaugh, William R. and Feinstein David L. (Editors) "*IS'94 The DPMA Two Year Model Curriculum for IS Professionals*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1994.

[Madsen et al., 1993] Madsen, Ole Lehrmann, Møller-Pedersen, Birger, Nygaard, Kristen. "*Object-Oriented Programming in the BETA Programming Language*". Addison-Wesley, Wokingham (U.K.), 1993.

[Manns, 1999] Manns, Mary Lynn. "*Pedagogical Pattern #8. Lab-Discussion-Lecture-Lab (LDLL) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp8.htm. [Última vez visitado, 20/8/1999]. July, 1999.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[Marqués, 1995] Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

[Meyer, 1992] Meyer, Bertrand. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

[Meyer, 1996] Meyer, Bertrand. "*Teaching Object Technology*". IEEE Computer, 29(12):117. December, 1996.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Miller and Mingins, 1998] Miller, Jan and Mingins, Christine. "*Putting the Practice into Software Education*". In Proceedings oh the 1998 International Conference on Software Engineering: Education and Practice (SEEP'98). (26-29 January 1998, Dunedin – New Zeland). Pages, 200-208. IEEE Computer Society. 1998.

[Mohedano, 1998] Mohedano, José Eduardo. "*Java en la Historia: El Azar Mueve el Mundo*". Sólo Programadores. Especial Monográfico Nº3: 6-10. Octubre, 1998.

[Naughton, 1996] Naughton, Patrick. "*The Java Handbook*". McGraw-Hill, 1996.

[Naur and Randell, 1969] Naur, P. and Randell, B. (Editors). "*Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968*". Brussels: Scientific Affairs Division, NATO. January, 1969.

[Norman, 1988] Norman, D. A. "*The Psychology of Everyday Things*". New York: Basic Books, Inc. 1988.

[Northrop, 1992] Northrop, Linda M. "*Finding an Educational Perspective for Object-Oriented Development*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 245-249. ACM. 1992.

[Nunamaker et al., 1982] Nunamaker, Jay F., Couger, J. D. and Davis, Gordon B. "*Information System Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs*". Communications of the ACM 25(11). November, 1982.

[Nygaard and Dahl, 1981] Nygaard, Kristen and Dahl, Ole-Johan. "*The Development of the SIMULA Language*". In *History of Programming Languages*, Richard L. Wexelblat editor. Pages 439-493. Academic Press, 1981.

[OMG, 1998] OMG. "*OMG Unified Modeling Language Specification. Version 1.2*". Object Management Group Inc. ftp://ftp.omg.org/pub/docs/ad/98-12-02-pdf. July, 1998.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Orr, 1977] Orr, K. T. "*Structured System Development*". Yourdon Press, 1977.

[Orr, 1981] Orr, K. T. "*Structured Requirements Definition*". Ken Orr & Associates, 1981.

[Osborne, 1992] Osborne, Martin. "*The Role of Object-Oriented Technology in the Undergraduate Computer Science Curriculum*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 303-308. ACM. 1992.

[Paepcke, 1993] Andreas Paepcke (editor). "*Object-Oriented Programming: The CLOS Perspective*". MIT Press, Cambridge (Mass.), 1993.

[Pancake, 1995] Pancake, Cherri M. "*The Promise and the Cost of Object Technology: A Five Years Forecast*". Communications of the ACM 38(10):32-49. October, 1995.

[Parnas, 1972] Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

[Parnas and Weiss, 1987] Parnas, David Lorge and Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4): 259-265, December 1987.

[Parrish et al., 1998] Parrish, A., Borie, R., Cordes, D., Dixon, B., Hale, D., Hale, J., Jackson, J. and Sharpe, S. "*Computer Engineering, Computer Science and Management Information Systems: Partners in a Unified Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education & Training – CSEET'98. (February 22-25, 1998. Atlanta, GA – USA). Pages 67-75. IEEE Computer Society. 1998.

[Paulk et al., 1993a] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth and Weber, Charles V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-

TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

[Paulk et al., 1993b] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth and Weber, Charles V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July, 1993.

[Pham, 1997] Pham, Binh. "*The Changing Curriculum of Computing and Information Technology in Australia*". In Proceedings of the second Australasian conference on Computer science education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. 1997.

[Piattini, 1994] Piattini Velthuis, Mario G. "*Definición de una Metodología para el Desarrollo de Bases de Datos Orientadas al Objeto Fundamentadas en Extensiones del Modelo Relacional*". Tesis Doctoral. Facultad de Informática, Universidad Politécnica de Madrid. 1994.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

[Pressman, 1987] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 2nd edition. McGraw Hill, 1987.

[Pressman, 1992] Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Prieto and Victory, 1999] Prieto, Máximo and Victory, Pablo. "*Pedagogical Pattern #20. Identity Pattern*". Version 1.0. http://www-lifia.info.unlp.edu.ar/ppp/pp20.htm. [Última vez visitado, 20/8/1999]. July, 1999.

[Proto-Patterns, 1999] "*The Pedagogical Patterns Project. Successes in Teaching Object-Technology (PROTO-PATTERNS)*". http://www-lifia.info.unlp.edu.ar/ppp/index.html. [Última vez visitado, 20/8/1999]. July, 1999.

[Ramamoorthy and Sheu, 1988] Ramamoorthy, C. and Sheu, P. "*Object-Oriented Systems*". IEEE Expert, 3(3). Fall, 1988.

[Rand, 1979] Rand, Ayn. "*Introduction to Objectivist Epistemology*". New American Library, 1979.

[Rans, 1999] Rans, Michael. "*A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development*". http://users.ox.ac.uk/~ball0370/documents/oo.pdf [Última vez visitado, 22/12/1999]. November, 1999.

[Randell, 1998] Randell, Brian. "*Memories of the NATO Software Engineering Conference*". In *Anecdotes Column*, James E. Tomayko (Editor). IEEE Annals of the History of Computing, 20(1):51-54. January-March, 1998.

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Redwine, 1985] Redwine, S. T. "*Software Technology Maduration*". In Proceedings of the 1st International Conference on Software Engineering, Washington D. C. (USA). IEEE, 1985.

[Reenskaug et al., 1996] Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

[Reynolds and Fox, 1996] Reynolds, Charles and Fox, Christopher. "*Requirements for a Computer Science Curriculum Emphasizing Information Technology Subject Area: Curriculum Issues*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 247-251. ACM. 1996.

[Rivas et al., 1997] Rivas, Erick, DeSilva, Dilhar, McDaniel, Terrie and Atkinson, Colin. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January, 1997.

[Roberts et al., 1999] Roberts, Eric, Shackelford, Russ, LeBlanc, Rich and Denning, Peter J. "*Curriculum 2001: Interim Report from the ACM/IEEE-CS Task Force*". In Proceedings of the thirtieth SIGCSE technical symposium on Computer science education, SIGCSE '99. (March 24-28, 1999, New Orleans, LA - USA). Pages 343-344. ACM. 1999.

[Rosson and Carroll, 1996] Rosson, Mary Beth and Carroll, John M. "*Scaffolded Examples for Learning Object-Oriented Design*". Communications of the ACM, 39(4):46-47. April, 1996.

[Rumbaugh, 1996] Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1999] Rumbaugh, James, Jacobson, Ivar and Booch, Grady. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

[Sawyer and Kotonya, 1999] Sawyer, Pete and Kotonya, Gerald. "*SWEBOK: Software Requirements Engineering Knowledge Area Description*". In [Abran et al., 1999], 1999.

[Schmauch, 1994] Schmauch, C. "*ISO9000 for Software Developers*". IEEE Computer Society Press, 1994.

[Schmucker, 1986] Schmucker, K. "*Object-Oriented Language for the Macintosh*". Byte, 11(8). August, 1986.

[Scragg et al., 1994] Scragg, Greg, Baldwin, Doug and Koomen, Hans. "*Computer Science Needs and Insight-Based Curriculum*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 150-154. ACM. 1994.

[Sernadas et al., 1989] Sernadas, A., Fiadeiro, J., Sernadas, C. and Eric, H.-D. "*The Basic Building Blocks of Information Systems*". In *Information Systems Concepts*. North Holland, Namur, 1989.

[Shackelford and LeBlanc, 1994] Shackelford, Russell L. and LeBlanc, Richard J. "*Integrating 'Depth First' and 'Breadth First' Models of Computing Curricula*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science

education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 6-10. ACM. 1994.

[Sharp et al., 2000] Sharp, Helen, Robinson, Hugh and Woodman, Mark. "*Software Engineering: Community and Culture*". IEEE Software, 17(1):40-47. January/February, 2000.

[Shaw, 1984] Shaw, Mary. "*Abstraction Techniques in Modern Programming Languages*". IEEE Software, 1(4). October, 1984.

[Shaw, 1985] Shaw, Mary (editor). "*The Carnegie-Mellon Curriculum for Undergraduate Computer Science*". Springer-Verlag, 1985.

[Shaw, 1990] Shaw, Mary. "*Prospects for an Engineering Discipline of Software*". IEEE Software, 7(6):15-24. November, 1990.

[Shaw and Garlan, 1996] Shaw, Mary and Garlan, David. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996.

[Shaw and Tomayko, 1991] Shaw, Mary and Tomayko, James E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August, 1991.

[Shlaer and Mellor, 1992] Shlaer, S. and Mellor, S. "*Object Lifecycles: Modeling the World in States*". Yourdon Press, 1992.

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

[Smith and Tockey, 1988] Smith, M. and Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

[Sommerville, 1985] Sommerville, Ian. "*Software Engineering*". 2nd edition. Addison-Wesley, 1985.

[Sommerville, 1989] Sommerville, Ian. "*Software Engineering*". 3rd edition. Addison-Wesley, 1989.

[Sommerville, 1996] Sommerville, Ian. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

[Spivey, 1989] Spivey, J. M. "*The Z Notation. A Reference Manual*". International Series in Computer Science. Prentice-Hall International, 1989.

[Stillings et al., 1987] Stillings, N., Feinstein, M., Garfield, J., Rissland, E., Rosenbaum, D., Weisler, S. and Baker-Ward, L. "*Cognitive Science: An Introduction*". The MIT Press, 1987.

[Stroustrup, 1986] Stroustrup, Bjarne. "*The C++ Programming Language*". 1st Edition, Addison Wesley, 1986.

[Stroustrup, 1986b] Stroustrup, Bjarne. "*What Is Object-Oriented Programming?*". In Proceedings of 14th ASU Conference. August 1986.

[Stroustrup, 1991] Stroustrup, Bjarne. "*The C++ Programming Language*". 2nd Edition, Addison Wesley, 1991.

[Stroustrup, 1994] Stroustrup, Bjarne. "*The Design and Evolution of C++*". Addison Wesley, 1994.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

[SUN, 2000] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

[Sutcliffe and Maiden, 1998] Sutcliffe, Alistair and Maiden, Neil. "*The Domain Theory for Requirements Engineering*". IEEE Transactions on Software Engineering, 24(3): 174-196. March, 1998.

[Tewari and Friedman, 1992] Tewari, Raj and Friedman, Frank. "*The Impact of Object-Oriented Software Engineering in the Introductory Computer Science Curriculum*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 289-292. ACM. 1992.

[Tomayko, 1987] Tomayko, James E. "*Teaching a Project-Intensive Introduction to Software Engineering*". Technical Report CMU/SEI-87-TR-20 (ESD-TR-87-171), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August, 1987.

[Tomayko, 2000] Tomayko, James E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

[Tremblay, 1999] Tremblay, Guy. "*Knowledge Area Description for Design (version 0.5)*". In [Abran et al., 1999], 1999.

[Tucker and Wegner, 1994] Tucker, Allen B. and Wegner, Peter. "*New Directions in the Introductory Computer Science Curriculum*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 11-15. ACM. 1994.

[Tucker et al., 1991a] Tucker, Allen B., Barnes, Bruce H., Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke, Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*Computing Curricula 1991*". ACM Press. February, 1991.

[Tucker et al., 1991b] Tucker, Allen B., Barnes, Bruce H., Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke, Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report. Computing Curricula 1991*". Communications of the ACM, 34(6):68-84. June, 1991.

[Tucker et al., 1996] Tucker, Allen B., Astrachan, Owen, Bruce, Kim, Cupper, Robert, Denning, Peter, Drysdale, Scot, Horton, Tom, Kelemen, Charles, McGeoch, Cathy, Patt, Yale, Proulx, Viera, Rada, Roy, Rasala, Richard, Roberts, Eric, Rudich, Steven, Stein, Lynn, Van Loan, Charles. "*Strategic Directions in Computer Science Education*". ACM Computing Surveys, 28(4):836-845. December, 1996.

[Tymann et al., 1994] Tymann, Paul T., Lea, Douglas and Raj, Rajendra K. "*Developing an Undergraduate Software Engineering Program in a Liberal Arts College*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science

Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 276-280. ACM. 1994.

[Ungar et al., 1992] Ungar, David, Smith, Randall B., Chambers, Craig and Hölzle, Urs. "*Object, Message and Performance: How they Coexist in Self*". IEEE Computer 25(10): 53-64. October, 1992.

[USAL, 1998] Universidad de Salamanca. "*Guía Académica Curso 1998-1999. Facultad de Ciencias*". Facultad de Ciencias - Universidad de Salamanca. 1998.

[USAL, 1999] Universidad de Salamanca. "*Guía Académica Curso 1999-2000. Facultad de Ciencias*". Facultad de Ciencias - Universidad de Salamanca. 1999.

[Vaitkevitchius, 1999] Vaitkevitchius, Raimundas. "*Pedagogical Pattern #25. BASE-and-Supplementary-Languages in Lectures (BSLL)*". In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp25.htm. [Última vez visitado, 20/8/1999]. July, 1999.

[Vaughn, 2000] Vaughn, Rayford B. Jr. "*Software Engineering Degree Programs*". Crosstalk, The Journal of Defense Software Engineering, 13(3):7-9. March, 2000.

[Walker and Schneider, 1996] Walker, Henry M. and Schneider, G. Michael. "*A Revised Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 39(12):85-95. December, 1996.

[Wand, 1989] Wand, Y. "*How to Integrate Object Orientation with Structured Analysis and Design*". IEEE Software, 6(2). March, 1989.

[Ward and Mellor, 1985] Ward, P. and Mellor, S. "*Structured Development for Real-Time Systems*". Vols. 1-3. Yourdon Press, 1985.

[Warnier, 1974] Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

[Wegner, 1990] Wegner, Peter. "*Concepts and Paradigms of Object-Oriented Programming*". OOPS Messenger, 1(1). August, 1990.

[Weinberg, 1971] Weinberg, G. F. "*The Psychology of Computer Programming*". Van Nostrand Reinhold, 1971.

[Wielinga et al. 1991] Wielinga, B. J., Schreiber, A. T. and Breuker, J. A. "*KADS: A Modeling Approach to Knowledge Engineering*". Technical Report ESPRIT Project P5248 KAD-II, 1991.

[Wirfs-Brock and Johnson, 1990] Wirfs-Brock, Rebecca and Johnson, Ralph E. "*Surveying Current Research in Object-Oriented Design*". Communications of the ACM, 33(9):104-124. September, 1990.

[Wirfs-Brock et al., 1990] Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall. 1990.

[Wirth and Reiser, 1992] Wirth, Niklaus and Reiser, Martin. "*Programming in Oberon — Steps Beyond Pascal and Modula*". Addison-Wesley, Reading (Mass.), 1992.

[Woodman et al., 1996] Woodman, Mark, Davies, Gordon and Holland, Simon. "*The Joy of Software – Starting with Objects*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 88-92. ACM. 1996.

[X3J16/WG21, 1996] ANSI X3J16 and ISO WG21. "*Working Paper for Draft Proposed International Standard for Information Systems—Programming Language C++*".

ftp://research.att.com/dist/c++std/WP/CD2. [Última vez visitado 7/1/2000]. X3J16/96–0225 (WG21/N1043). December, 1996.

[Yonezawa and Tokoro, 1987] Yonezawa, A. and Tokoro, M. "*Object-Oriented Concurrent Programming: An Introduction*". In *Object-Oriented Concurrent Programming*. Cambridge, MA: The MIT Press. 1987.

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Yourdon Inc., 1993] Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

[Yourdon and Constantine, 1975] Yordon, E. and Constantine, L. "*Structured Design*". 1st edition. Prentice Hall, 1975.

[Yourdon and Constantine, 1979] Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

[Yourdon and Constantine, 1989] Yordon, E. and Constantine, L. "*Structured Design*". 2nd edition. Yourdon Press, 1989.

Capítulo 5
Programación Docente

Tras haber repasado el contexto de la Ingeniería del Software y de la Programación Orientada a Objetos, tanto en el plano histórico, como en el conceptual y el educativo, se van a detallar los objetivos y los programas de las dos asignaturas que, en el contexto de la Universidad de Salamanca, se ajustan al perfil de la plaza objeto de concurso.

En el diseño de los programas correspondientes a estas asignaturas se han tenido en cuenta, además de las influencias de los diferentes cuerpos de conocimiento y currículos, las siguientes consideraciones:

- Primeramente, debe tenerse presente que el descriptor general de estas asignaturas señala como contenidos los siguientes: *Planificación y gestión de proyectos informáticos. Análisis de requisitos de aplicaciones informáticas. Diseño y mantenimiento de aplicaciones informáticas*.
- La formación en temas como lenguajes de programación, algoritmos, bases de datos, pruebas, interfaces de usuario, arquitectura de ordenadores, rendimientos de sistema o sistemas operativos, está asignada a otras asignaturas del vigente plan de estudios.
- La organización docente de las asignaturas se realiza atendiendo a las dos dimensiones proceso/producto. En la parte de proceso los contenidos se centran principalmente en la actividad de desarrollo y en los aspectos de abstracción, representación, métodos y herramientas. La parte de producto se enfoca desde la realización de un proyecto, realizado en grupo, y con una temática que puede estar relacionada con los sistemas en tiempo real o con la gestión.
- Tanto los métodos de análisis y diseño estructurado como los orientados a objetos tienen cabida en la asignatura de Ingeniería del Software.

- Por su parte la asignatura de Programación Orientada a Objetos, se centra especialmente en la parte de diseño y programación, utilizando un lenguaje de programación orientado a objetos no como un fin en la asignatura, sino como un medio para afianzar los conceptos presentados.
- El trabajo en equipo es una constante en la parte práctica de estas asignaturas, buscando que el alumno se acostumbre a formar parte de un equipo, a semejanza de como, probablemente, deberá desempañar su vida profesional.
- Se busca que las asignaturas contribuyan a la formación del alumno más allá de los contenidos técnicos, potenciando su capacidad de comunicación, tanto de forma oral como escrita.
- Estas asignaturas corresponden a una formación para estudiantes en Ingeniería Técnica, es decir, estudiantes no graduados, donde la formación debe orientarse a los conceptos básicos y fundamentales, con una fuerte componente práctica. Los alumnos que cursen estas asignaturas deben recibir una formación completa de primer ciclo, con independencia de su continuidad en un segundo ciclo.

5.1 Objetivos del Programa Docente

Los objetivos del programa propuesto se dividen en dos grupos: *objetivos generales* y *objetivos específicos*.

5.1.1 Objetivos generales

Los objetivos generales que se formulan son válidos para las dos asignaturas, por corresponder a lo que se entiende por objetivo en la formación en Ingeniería del Software en el contexto para el que se elabora el presente Proyecto Docente.

El **objetivo global** es producir profesionales que puedan resolver de forma sistemática y ordenada la producción de software de calidad, que respondan a las necesidades y exigencias de las organizaciones, además de ser capaces de evaluar las nuevas tecnologías o comprender cómo pueden aplicarse de la mejor forma posible a la práctica del desarrollo del software.

El estudiante al finalizar esta formación debe ser capaz de:

- *Identificar y establecer las fases y etapas que constituyen el desarrollo de un sistema de información.*
- *Identificar y modelar los requisitos del nuevo sistema a construir.*

Se procurará potenciar en los alumnos las capacidades para:

- "*Aprender a aprender*" y "*aprender a pensar*".
- *Desarrollar un espíritu crítico y una actitud abierta ante cambios de todo tipo que afecten a la sociedad en la que vive y en especial a los cambios científico-técnicos de su especialidad.*
- *Fomentar actitudes y adquirir técnicas para un eficaz trabajo en equipo.*
- *Desarrollar actitudes de curiosidad intelectual y rigor científico.*
- *Basar en criterios deontológicos su futuro comportamiento en el ejercicio de la profesión.*
- *Estimular el perfeccionamiento profesional y la ampliación de estudios.*

Para conseguir estos objetivos se propone un programa diseñado para proporcionar al estudiante un cuerpo de conocimiento - que incluya la cobertura de las actividades y herramientas del proceso de desarrollo de proyectos, sus aspectos y los productos que elabora - y una experiencia en el desarrollo de un sistema software que les permita aplicar los conocimientos adquiridos en las clases teóricas.

5.1.2 Objetivos específicos

Los objetivos específicos de las asignaturas aquí presentadas se obtienen de los objetivos establecidos en la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca [García et al., 2000], que se catalogan en tres apartados: *conceptos teóricos, aspectos prácticos* y *habilidades personales*.

Los objetivos de la unidad docente en el apartado de los conceptos teóricos son:

T1 Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.

T2 Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.

T5 Especificaciones formales de requisitos.

T6 Método de análisis/diseño estructurado.

T7 Método de análisis/diseño orientado a objetos.

T8 Diseño de la interfaz gráfica de usuario.

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

T10 Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

T11 Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

T13 Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

Los objetivos de la unidad docente en el apartado de los aspectos prácticos son:

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de la Orientación a Objetos.

P3 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

P5 Programación orientada a objetos.

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

P7 Aprendizaje y manejo de forma práctica de plataformas, entornos de desarrollo, lenguajes de programación... de alta repercusión en el desarrollo de sistemas software en la actualidad.

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

P9 Construcción de sistemas software de entidad superior a una práctica de laboratorio, a ser posible partiendo de unas especificaciones reales obtenidas de *clientes* y/o *usuarios* reales.

Los objetivos de la unidad docente en el apartado de habilidades personales son:

H1 Mejora de la expresión oral.

H2 Mejora en la redacción de documentos técnicos.

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

H4 Capacitar a los alumnos para el trabajo en grupo.

La unidad docente de Ingeniería del Software y Orientación a Objetos se ha creado para dar cobertura a las titulaciones de Ingeniería Técnica en Informática de Sistemas (I.T.I.S) (Plan de 1997) e Ingeniero en Informática (I.I) (Plan de 1998) impartidas en la Universidad de Salamanca.

En estas titulaciones las asignaturas que mejor se ajustan a los objetivos marcados en la unidad docente se recogen en la Tabla 5.1. Dentro de estas asignaturas se encuentran las dos objeto de este Proyecto Docente, siendo el resto aquéllas con las que tienen una relación más estrecha.

Las dependencias e interrelaciones entre estas asignaturas se muestran en la Figura 5.1. En el establecimiento de estas dependencias se han tenido en cuenta el factor tiempo, que claramente establece el orden lógico en el que se van a cursar las asignaturas, como las aportaciones a la unidad docente de las mismas, en forma de contenidos y objetivos a conseguir.

Además de las asignaturas propias de la unidad docente, se incluye la asignatura de Diseño de Bases de Datos. Esto se debe a que es la asignatura en la que se introducen los modelos de datos conceptuales y lógicos (diagramas entidad/relación y modelos relacionales típicamente), lo que supone una importante base, a la vez que una descarga, para la asignatura de Ingeniería del Software donde estos modelos serán utilizados de forma práctica sin necesidad de tener que incluirlos en la parte teórica de la asignatura.

Asignatura	Titulación	Curso	Carácter	Créditos
Interfaces Gráficas	I.T.I.S	2º	Optativa	6 (3T + 3P)
Ingeniería del Software	**I.T.I.S**	**3º**	**Obligatoria**	**6 (4,5T + 1,5P)**
Programación Orientada a Objetos	**I.T.I.S**	**3º**	**Optativa**	**6 (3T + 3P)**
Proyecto	I.T.I.S	3º	Obligatoria	9 (9P)
Análisis de Sistemas	I.I	1º	Troncal	9 (6T + 3P)
Administración de Proyectos Informáticos	I.I	2º	Troncal	9 (6T + 3P)
Sistemas de Información	I.I	2º	Troncal	9 (9P)
Proyecto	I.I	2º	Troncal	6 (6P)

Tabla 5.1. Asignaturas que componen la unidad docente

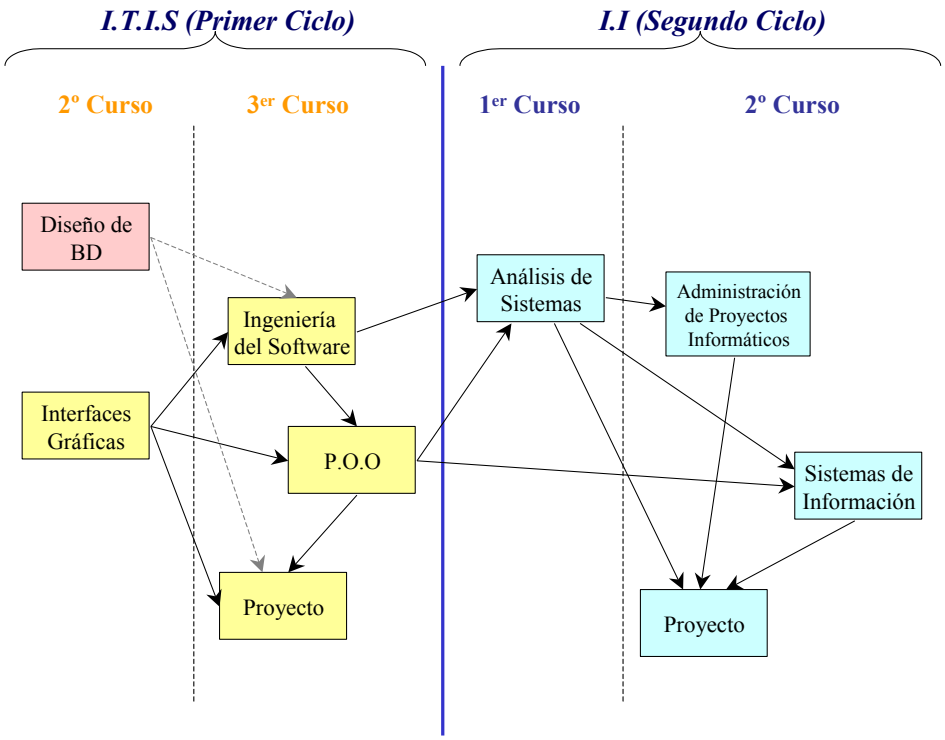

Figura 5.1. Dependencias entre las asignaturas que forman la unidad docente

Los objetivos identificados se reparten en las asignaturas de la unidad docente como se indica en la Tabla 5.2, señalando de una manera especial aquéllos que son cometido de las asignaturas objeto de este Proyecto Docente.

	Obj. Teóricos	Obj. Prácticos	Hab. Personales
Interfaces Gráficas	T8	P6	H1, H2, H3
Ingeniería del Software	**T1, T2, T3, T4, T6, T7, T9**	**P1, P2, P4**	**H1, H2, H3, H4**
Programación Orientada a Objetos	**T7, T9**	**P2, P5**	**H1, H2, H3, H4**
Proyecto I.T.I.S		P9	H1, H2, H3
Análisis de Sistemas	T3, T4, T5, T7, T12, T13	P1, P2, P4	H1, H2, H3, H4
Administración de Proyectos Informáticos	T10, T11, T12	P4, P8	H1, H2, H3, H4
Sistemas de Información		P7	H1, H2, H3
Proyecto I.I		P8, P9	H1, H2, H3

Tabla 5.2. Reparto de los requisitos en las asignaturas de la unidad docente

5.2 Programa de la asignatura Ingeniería del Software

La asignatura de *Ingeniería del Software* es la clave de la formación de los alumnos de la *Ingeniería Técnica en Informática de Sistemas* en materias de Ingeniería del Software.

Asignatura	*Ingeniería del Software (obligatoria)*
Créditos	*4,5T + 1,5P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*1º*
Responsable	*Francisco José García Peñalvo (fgarcia@gugu.usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla 5.3. Ficha de la asignatura Ingeniería del Software

Como ya se ha comentado previamente en este Proyecto Docente, esta asignatura se imparte en el quinto cuatrimestre (o lo que es lo mismo, en el primer cuatrimestre del tercer y último curso de la titulación) dentro del plan de estudios del B.O.E de 4-11-1997.

Esta asignatura está dotada de ***6 créditos***, ***4,5 teóricos*** y ***1,5 prácticos***, lo que conduce al principal problema de la misma: *un número escaso de créditos para la cantidad de materia que puede tratarse bajo este epígrafe*. Este problema se aborda minimizando los contenidos que se solapan con otras asignaturas de la titulación, descargando temas *avanzados* en las asignaturas afines del segundo ciclo y enfocando la asignatura de *Programación Orientada a Objetos* como una continuación de la asignatura de *Ingeniería del Software*, aunque centrada en los aspectos de diseño e implementación dentro del paradigma objetual.

En la elaboración del programa se ha optado por una estrategia cuyos ejes básicos son:

1. Mantener los temas fundamentales que tradicionalmente se imparten en este tipo de asignaturas en otras Universidades, tanto españolas como extranjeras, así como los derivados de las recomendaciones de los currículos internacionales y los diferentes cuerpos de conocimiento de la Ingeniería del Software.
2. Se parte de los siguientes prerrequisitos:
    a. El alumno debe estar familiarizado con la teoría y la práctica del diseño y codificación en lenguajes procedurales (*por ejemplo C [Kernighan and Ritchie, 1988]*). Estos conocimientos deben adquirirse en las asignaturas de relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos**.
    b. El alumno debe tener los conocimientos y la práctica necesaria en la creación de modelos de información conceptuales y lógicos, en concreto, dominio del modelo entidad-relación, paso al modelo relacional y normalización. Estos conceptos se adquieren en la asignatura de segundo curso **Diseño de Bases de Datos** (*asignatura troncal en el plan de estudios vigente*).
    c. Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

3. No dejar que en ningún caso el alumno pierda de vista que el desarrollo de todo sistema software debe abordarse con un proceso sistemático que englobe los procedimientos, las técnicas y las herramientas necesarias.
4. Hacer hueco a las, cada día más demandadas, técnicas de desarrollo orientadas a objetos, dentro de un currículo dominado por el paradigma estructurado.
5. Dentro de los procesos del ciclo de vida, esta asignatura se decanta por los procesos de desarrollo, y dentro de éstos por las actividades de análisis de requisitos y diseño.
6. Aunque la asignatura se fundamenta en la transmisión de conocimiento técnico, no se puede (*ni se quiere*) perder la oportunidad de hacer que los alumnos desarrollen y potencien otras habilidades más generales, pero de importancia capital en su futuro como profesionales: *acostumbrarse a consultar bibliografía (especialmente en inglés), haciendo hincapié en la importancia que tiene leer con cuidado para sintetizar, escribir y modelar de forma adecuada* [Jackson, 1995]; *escribir documentos técnicos que describan los diferentes elementos software que se crean a lo largo de un proyecto* [Deveaux et al., 1999] *cuidando los estándares de documentación* [Gersting, 1994], [McCauley et al., 1996]*, sin que ello signifique dar la espalda a las reglas gramaticales y de estilo que ofrece un lenguaje tan rico como el castellano* [Vaquero, 1999]; *desarrollar una capacidad de comunicación oral adecuada* [McDonald and McDonald, 1993], [Fell et al., 1996].
7. Llegar a un equilibrio entre la teoría y la práctica [Glass, 1996], de forma que una base teórica bien establecida sea el fundamento adecuado para la aplicación práctica de la Ingeniería del Software.

5.2.1 Programa de la parte teórica

La parte teórica de la asignatura de Ingeniería del Software está orientada para satisfacer aquellos objetivos teóricos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**T1, T2, T3, T4, T6, T7** y **T9**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000] (*incluido en el Apéndice A de este Proyecto Docente*).

T1	Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
T2	Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.
T3	Importancia de los requisitos en el ciclo de vida del software.
T4	Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
T6	Método de análisis/diseño estructurado.
T7	Método de análisis/diseño orientado a objetos.
T9	Estudio y comprensión de los fundamentos del diseño de sistemas software.

Tabla 5.4. Objetivos del programa de teoría de la asignatura Ingeniería del Software

5.2.1.1 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de nueve temas, organizados en cuatro unidades docentes, como se muestra en la Tabla 5.5.

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (6 Horas)
Tema 1. Introducción a la Ingeniería del Software (6 Horas)
Unidad Docente II: Paradigma estructurado de desarrollo (23 Horas)
Tema 2. Análisis y especificación de requisitos (11 Horas)
Tema 3. Análisis estructurado (2 Horas)
Tema 4. Diseño del software (6 Horas)
Tema 5. Diseño estructurado (4 Horas)
Unidad Docente III: Introducción al paradigma objetual (13 Horas)
Tema 6. Introducción a la Orientación a Objetos (6 Horas)
Tema 7. UML (6 Horas)
Tema 8. Visión general de la metodología OMT (1 Hora)
Unidad Docente IV: Miscelánea (2 Horas)
Tema 9. Herramientas CASE (2 Horas)

Tabla 5.5. Estructura del programa de teoría de la asignatura Ingeniería del Software (4,5 créditos)

La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve exposición de lo que es la Ingeniería del Software y la visión que se pretende conseguir desde la asignatura. Para que esto sea efectivo los alumnos deben contar con el programa de la asignatura, resumido en la guía académica que se les

entrega con la matrícula, y totalmente actualizado en la página web de la asignatura (*http://tejo.usal.es/~fgarcia/docencia.html*).

En la Figura 5.3 y en la Figura 5.4 se puede apreciar como se reparten las horas del temario teórico de esta asignatura en sus unidades docentes y temas.

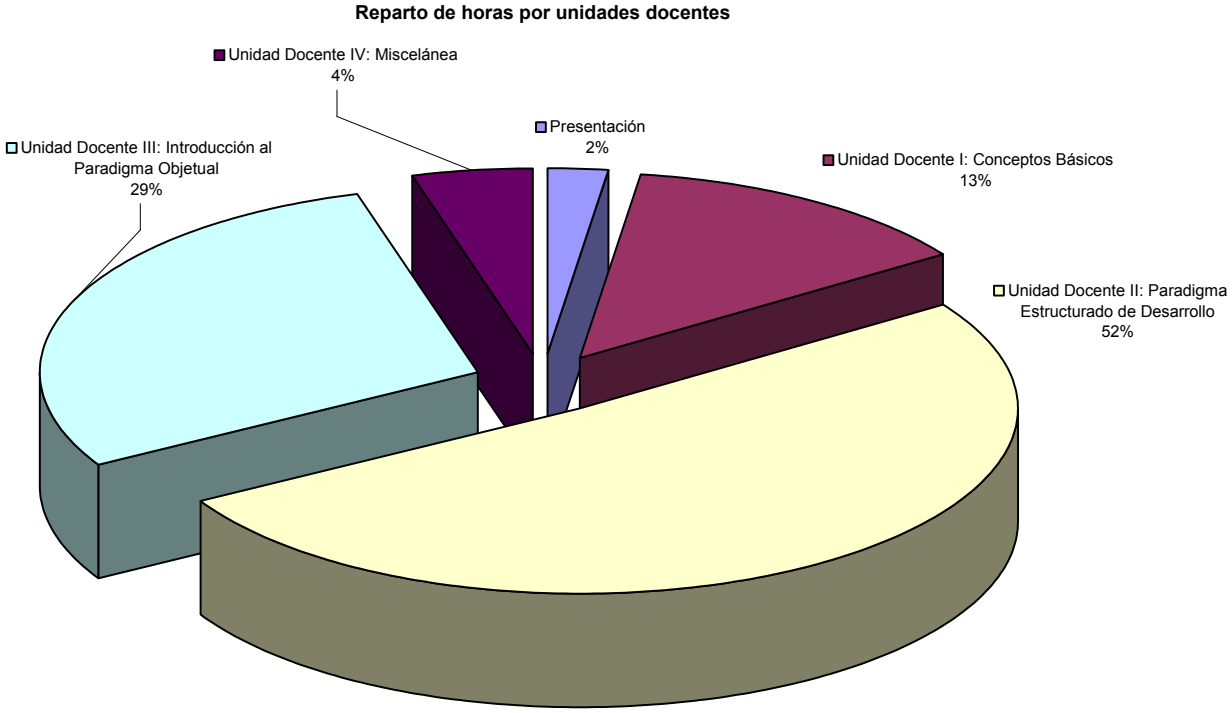

Figura 5.2. Reparto de las horas de teoría de Ingeniería del Software entre las unidades docentes

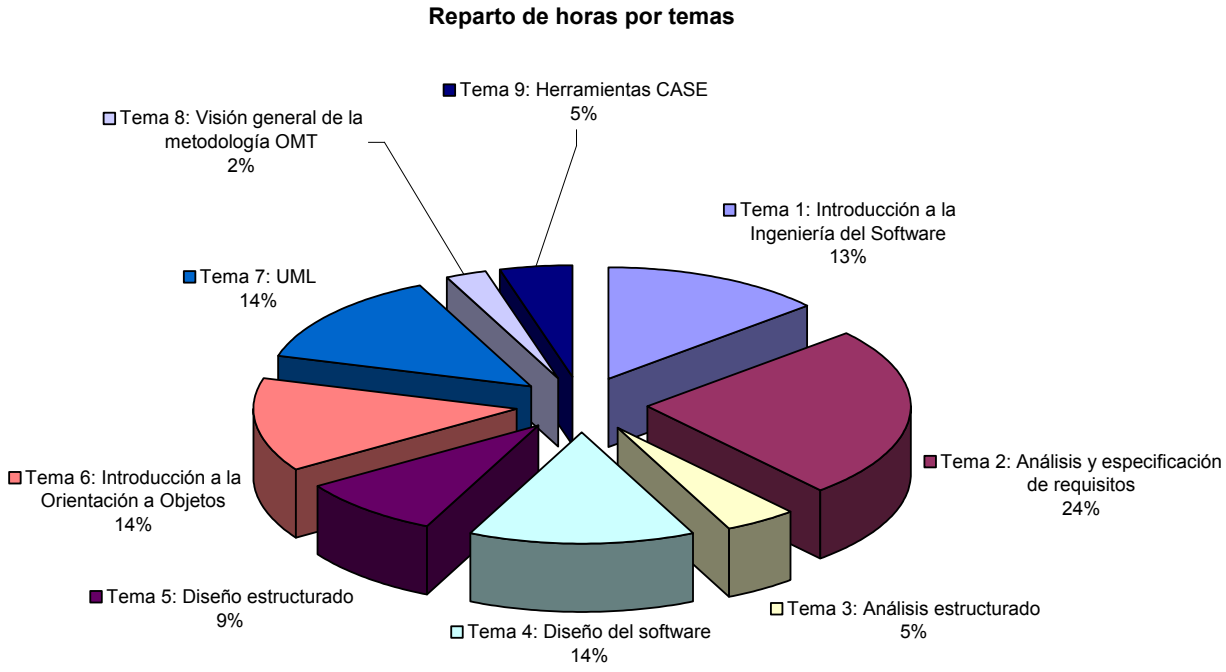

Figura 5.3. Reparto de las horas de teoría de Ingeniería del Software entre los temas

En la Tabla 5.6 se presenta la correspondencia existente entre el programa de teoría y los objetivos teóricos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T1, T2, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T9, H3
Tema 5	T6, H3
Tema 6	T3, T7, H3
Tema 7	T7, H3
Tema 8	T7, H3
Tema 9	*P4*, H3

Tabla 5.6. Correspondencia entre el temario teórico y los objetivos teóricos de la asignatura

La **Unidad Docente IV**: ***Miscelánea***, es difícil de impartir en la realidad por limitaciones de tiempo. El **Tema 9**, dedicado a las herramientas CASE, es factible introducirlo a la vez que se realiza la **Práctica 4**. Otros posibles temas candidatos a ser tratados en esta unidad miscelánea, podrían ser: *pruebas del software, mantenimiento, reingeniería, calidad del software, gestión de proyectos...*

Estos y otros temas pueden ser objeto de actividades complementarias, que se llevarán a cabo dependiendo de diversos factores, entre los que cabe citar *la disponibilidad de tiempo para hacerlas efectivas* y *el interés y colaboración de los propios alumnos*. Las actividades complementarias a realizar pueden caer dentro de alguno de los siguientes grupos:

- *Seminarios impartidos sobre temas específicos.*
- *Trabajos voluntarios realizados por los alumnos.*
- *Conferencias invitadas.*
- *Workshop de trabajos realizados por los alumnos sobre temas de ingeniería del software y objetos.*

Desarrollo de las clases de teoría

Las clases de teoría se llevan a cabo utilizando una variante de la clase magistral, donde el profesor se apoya en un retroproyector y en la pizarra para el desarrollo de los nueve temas de que se compone el temario.

Los alumnos cuentan de antemano con las transparencias de los temas para que no tengan que tomar apuntes en el sentido clásico del término, y puedan prestar atención a las explicaciones, completando las transparencias con las notas que cada uno crea oportuno. Además, permite que el alumno que lo desee intervenga en cualquier

momento para hacer una pregunta o solventar una duda y no, como en el dictado de apuntes, para pedir que se repita una frase.

El uso que de las transparencias hace el que suscribe este Proyecto Docente, es el de guión de clase. De forma, que recojan los puntos fundamentales que se van a desarrollar, más esquemas, gráficos y definiciones de términos. Esto obliga a que la clase no se convierta en una mera lectura de las transparencias, y se asemeje más a una conferencia centrada en el tema abordado en cada clase.

Esta forma de abordar las clases teóricas tiene la ventaja de que obliga al alumno interesado por la asignatura a prestar atención a las explicaciones y a completar el material distribuido por el profesor con la bibliografía de consulta básica o las lecturas complementarias que se recomiendan a lo largo de cada uno de los temas (por este motivo, aunque hay unos apuntes desarrollados por el profesor [García, 1999], éstos no le son facilitados a los alumnos).

La elección de este forma de impartir clases viene *forzada* por dos factores fundamentales:

- *La infraestructura con la que se cuenta*. Una clase con capacidad para recoger los alumnos matriculados, cuyo número ha rondado los dos últimos cursos la cifra de 140 alumnos.
- *La gran cantidad de conceptos que se han de impartir*. Es la primera asignatura que, en su plan de estudios, afronta el desarrollo de software desde una perspectiva global a través de su ciclo de vida, y no centrada en los algoritmos, las bases de datos o las redes por ejemplo.

Las ventajas que se obtienen con este método docente se pueden resumir en los siguientes puntos:

- Permite al docente controlar el *tiempo de la clase*, siendo lo suficientemente flexible para que el profesor pueda sortear los posibles imprevistos en la planificación del calendario, haciendo hincapié en los conceptos más importantes y pasando más someramente por los temas más sencillos o de menor trascendencia.
- Requiere una atención constante del alumno, para seguir las explicaciones del profesor.
- Obliga al alumno a consultar la bibliografía y las lecturas recomendadas para completar el material facilitado por el profesor, orientado a servir de base para las clases, pero a todas luces insuficiente para preparar la asignatura.

Como inconvenientes cabe citar:

- Obliga al profesor a realizar diferentes cambios de ritmo, de tono... para no caer en la monotonía del pasar transparencias.

- Si el alumno *desconecta* de las explicaciones no sacará provecho de sus asistencia a clase, lo cual es muy factible por el relax que supone no tener que copiar apuntes.
- Si el alumno confunde las transparencias facilitadas con *toda la verdad sobre la asignatura,* estará en desventaja para preparar y comprender la asignatura.

Evaluación de la parte teórica

La forma principal de evaluar la parte teórica de esta asignatura es mediante la realización de una prueba escrita. Aunque también se puede aumentar la nota final realizando trabajos teóricos o participando en seminarios (*aunque no incida esta nota para aprobar un examen suspenso, ya sea de teoría o de prácticas*).

En la Figura 5.4 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota del examen de teoría y los trabajos voluntarios en dicha nota.

Si (***Teoría*** ≥ 4,75) **y** (Práctica ≥ 5.0)
    Nota Final = (***Teoría****0,5) + (Práctica*0,5) + ***Nota trabajos***
Sino
    ∅
Fin si

Figura 5.4. Influencia de la nota en la parte teórica en la nota final de Ingeniería del Software

A la hora de elaborar el examen de la asignatura se tienen en cuenta los siguientes aspectos:

- El examen se divide en dos partes que hay que aprobar por separado para superar la prueba. La primera parte se centra en la evaluación de conceptos desarrollados en la asignatura, mientras que la segunda es un conjunto de supuestos prácticos, en la que aparecen algunas de las técnicas de modelado estudiadas.
- La primera parte del examen está compuesta por un conjunto de cuestiones de respuesta abierta y corta, donde se prima evaluar la capacidad de asimilación, de comprensión y de relacionar los conceptos desarrollados en la asignatura, sobre la capacidad memorística del alumno.
- Los supuestos de la segunda parte buscan comprobar que los modelos realizados en la práctica obligatoria han sido asimilados por todos los integrantes del grupo, así como evaluar el dominio de otras técnicas de modelado menos utilizadas en las prácticas.

Bibliografía básica de referencia

Desde esta asignatura, así como en la otra que forma este Proyecto Docente, se entiende que el conocimiento es tanto el que se posee como el que se sabe conseguir. Por este motivo, se fomenta el manejo de libros y otros recursos bibliográficos.

Así, junto con el programa de la asignatura se le facilita a los alumnos una lista con la bibliografía básica que pueden (*y deben*) consultar para preparar y dominar la asignatura de Ingeniería del Software.

A la hora de elaborar esta lista se tienen en cuenta los siguientes factores:

- No se trata de distribuir un listado exhaustivo de todos los títulos que el profesor conoce o maneja para preparar sus clases, sino una lista representativa que trate todo el temario en global (*para los aspectos concretos ya se recomendarán lecturas complementarias en cada tema*).
- En la elección de los títulos influye su disponibilidad para los alumnos, bien en la biblioteca o a través el profesor.
- El idioma, de forma que si un libro está traducido al español prevalecerá sobre el original. Esto es debido a que, aunque los alumnos deben acostumbrarse al manejo de bibliografía en inglés, sienten una aversión generalizada por los textos que no están español.

La lista de títulos que se les propone como bibliografía básica de consulta es:

- **Booch, Grady, Rumbaugh, James and Jacobson, Ivar.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.
- **Piattini, M. G. y Daryanani, S. N.** "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma. 1995.
- **Piattini, M. G., Calvo-Manzano, J. A., Cervera, J. y Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.
- **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill. 1998.
- **OMG.** "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.
- **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.
- **Rumbaugh, J., Jacobson, I. and Booch, G.** "*The Unified Modeling Language Reference Manual*". Addison-Wesley. 1999.
- **Sommerville, I.** "*Software Engineering*". 5th Edition, Addison-Wesley. 1996.
- **Yourdon, E. and Constantine, L. L.** "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall. 1979.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

5.2.1.2 Desarrollo comentado del programa de teoría

En este epígrafe se va a desarrollar con mayor grado de detalle el programa de la parte teórica de la asignatura Ingeniería del Software. Este programa se ha dividido en cuatro partes o unidades docentes.

Una primera que introduce la terminología básica que se manejan en la asignatura, en especial el concepto de ciclo de vida.

Una segunda donde se desarrollan las características del paradigma de desarrollo estructurado, en concreto de las fases de análisis y diseño estructurado.

Una tercera, en la que se realiza una introducción a la Orientación a Objetos, centrada en el modelo de objetos, en UML y en una sucinta presentación de la metodología OMT, génesis de otras avanzadas.

Por último, una cuarta donde, dependiendo del tiempo de que se disponga se tratan brevemente otros temas relacionados con la Ingeniería del Software.

Es obvio que, debido a los créditos asignados a la asignatura, no es posible extenderse en todas las partes por igual. En algunos casos la explicación se verá reducida a una breve referencia o comentario, mientras que en otros, que se consideran más representativos del estado y de mayor actualidad, se profundiza en las explicaciones.

Cada parte esta dividida en una serie de temas, subtemas y epígrafes. Cada uno de los temas va a ser descrito siguiendo el siguiente patrón de documentación:

- **Título:** Indica de forma concisa la denominación comúnmente asignada al tema correspondiente.
- **Descriptores:** Consiste en la enumeración de una serie de palabras clave, que ayudan a conocer el contenido de los temas tratados. Representan el papel de *índices* que facilitan las búsquedas de los conceptos tratados en cada uno de los temas, o en varios temas.
- **Objetivos:** Indica los objetivos que se buscan con la inclusión en el programa de cada uno de los temas.
- **Contenido:** Se señalan los subtemas y eventualmente epígrafes que se tratan dentro del tema considerado.
- **Resumen:** Elabora un resumen explicativo, a modo de epítome, de los diferentes conceptos que se desarrollan en la unidad temática.
- **Bibliografía:** Se citan para cada uno de los temas las referencias bibliográficas, clasificadas en cada caso en tres categorías, a saber:
    - *Referencias citadas en las transparencias de la asignatura.*
    - *Referencias de lecturas complementarias sobre el tema.*
    - *Referencias específicas utilizadas por el profesor para preparar la explicación del tema.* Con independencia de las anteriormente citadas.

Unidad Docente I: Conceptos básicos (6 Horas)

Tema 1. Introducción a la Ingeniería del Software (6 Horas)

1.1 Software (20 Minutos)

1.1.1 Definición
1.1.2 Evolución de los costes del software
1.1.3 La crisis del software
1.1.4 Problemas del desarrollo del software
1.1.5 Causas de los males del software

1.2 Sistemas de información (20 Minutos)

1.2.1 Definición
1.2.2 Objetivos de un sistema de información
1.2.3 Elementos de un sistema de información
1.2.4 Estructura de un sistema de información

1.3 Ingeniería del Software (20 Minutos)

1.3.1 Definición
1.3.2 Elementos de la Ingeniería del Software
1.3.3 Objetivos de la Ingeniería del Software

1.4 Ciclo de vida del software (1 Hora)

1.4.1 Definición
1.4.2 Objetivos del ciclo de vida
1.4.3 Descripción general del ciclo de vida del software
1.4.4 Curva del ciclo de vida del proyecto

1.5 Paradigmas de la Ingeniería del Software (3 Horas)

1.5.1 Modelo primitivo
1.5.2 Modelo barroco
1.5.3 Modelo clásico
1.5.4 Modelo estructurado
1.5.5 Modelo de prototipos
1.5.6 Modelo evolutivo
1.5.7 Modelo incremental
1.5.8 Modelo espiral
1.5.9 Estándar ISO 12207
1.5.10 Ciclo de vida para desarrollos OO

1.6 Metodologías (1 Hora)

1.6.1 Introducción
1.6.2 Definición
1.6.3 ¿Qué cubren las metodologías?
1.6.4 Objetivos de las metodologías
1.6.5 Características deseables en una metodología
1.6.6 Metodología versus método
1.6.7 Clasificación

Unidad Docente II: Paradigma estructurado de desarrollo (23 Horas)

Tema 2. Análisis y especificación de requisitos (11 Horas)

2.1 Introducción al análisis (1 Hora)

- 2.1.1 Generalidades
- 2.1.2 Definición de análisis
- 2.1.3 Requisitos
- 2.1.4 Objetivo
- 2.1.5 Tareas
- 2.1.6 Los principios del análisis

2.2 Especificación de requisitos del software (30 Minutos)

- 2.2.1 Definición
- 2.2.2 Características de una E.R.S
- 2.2.3 Estructura de una E.R.S

2.3 Técnicas de especificación (30 Minutos)

- 2.3.1 Generalidades
- 2.3.2 Técnicas gráficas
- 2.3.3 Técnicas textuales
- 2.3.4 Marcos o plantillas
- 2.3.5 Enfoque de modelado

2.4 Modelado funcional (7 Horas)

- 2.4.1 Introducción
- 2.4.2 DFD
- 2.4.3 DFDs nivelados
- 2.4.4 Diccionario de datos
- 2.4.5 Especificación de procesos

2.5 Modelado de la información (10 Minutos)

- 2.5.1 Introducción
- 2.5.2 Diagrama de estructura de datos

2.6 Modelado del comportamiento (80 Minutos)

- 2.6.1 Introducción
- 2.6.2 Lista de eventos
- 2.6.3 Diagrama de transición de estados

2.7 Balanceo de modelos (30 Minutos)

- 2.7.1 Balanceo de modelos
- 2.7.2 Técnicas matriciales

Tema 3. Análisis estructurado (2 Horas)

3.1 Introducción (10 Minutos)

3.2 Enfoque de modelado clásico (20 Minutos)

- 3.2.1 Los cuatro modelos del sistema
- 3.2.2 Los problemas del enfoque clásico

3.3 Enfoque de modelado de Yourdon (90 Minutos)

3.3.1 Introducción

3.3.2 Modelo esencial

3.3.3 Modelo de implantación

Tema 4. Diseño del software (6 Horas)

4.1 Introducción (15 Minutos)

4.1.1 Importancia del diseño en el ciclo de vida de un producto

4.1.2 Diseño dentro de la Ingeniería del Software

4.1.3 Evolución del diseño

4.2 Proceso de diseño (20 Minutos)

4.2.1 Introducción

4.2.2 Definición de diseño

4.2.3 Presentación de las actividades del diseño

4.3 Actividades de diseño (40 Minutos)

4.3.1 Diseño arquitectónico

4.3.2 Diseño de los datos

4.3.3 Diseño de procedimientos

4.3.4 Diseño de la interfaz

4.4 Fundamentos de diseño (2 Horas)

4.4.1 Introducción

4.4.2 Abstracción

4.4.3 Refinamiento sucesivo

4.4.4 Modularidad

4.4.5 Arquitectura del software

4.4.6 Estructura del programa

4.4.7 Partición estructural

4.4.8 Estructura de datos

4.4.9 Procedimiento del software

4.4.10 Ocultación de la información

4.5 Diseño modular (2 Horas y 45 Minutos)

4.5.1 Introducción

4.5.2 Definición de módulo

4.5.3 Tamaño de los módulos

4.5.4 Criterios y reglas para la evaluación de la modularidad

4.5.5 Complejidad de los módulos

4.5.6 Independencia modular

Tema 5. Diseño estructurado (4 Horas)

5.1 Introducción (10 Minutos)

5.2 Diagrama de estructuras (50 Minutos)

5.3 Estrategias de diseño (3 Horas)

5.3.1 Análisis de transformación

5.3.2 Análisis de transacción

Unidad Docente III: Introducción al paradigma objetual (13 Horas)

Tema 6. Introducción a la Orientación a Objetos (6 Horas)

6.1 Introducción y evolución de la Orientación a Objetos (1H)

6.1.1 Situación actual
6.1.2 Áreas de aplicación
6.1.3 Evolución de la Orientación a Objetos
6.1.4 Ventajas y desventajas
6.1.5 Orientación a Objetos versus reutilización del software

6.2 Modelo objeto (4 Horas y 30 Minutos)

6.2.1 Tipos de paradigmas de programación
6.2.2 Definición de modelo objeto
6.2.3 Concepto de objeto
6.2.4 Concepto de clase
6.2.5 Abstracción
6.2.6 Encapsulamiento
6.2.7 Modularidad
6.2.8 Jerarquía
6.2.9 Mensajes
6.2.10 Tipos
6.2.11 Polimorfismo
6.2.12 Enlaces y asociaciones
6.2.13 Relaciones entre clases

6.3 Análisis y diseño orientados a objetos (30 Minutos)

6.3.1 Introducción
6.3.2 Análisis orientado a objetos
6.3.3 Diseño orientado a objetos

Tema 7. UML (6 Horas)

7.1 Introducción (30 Minutos)

7.1.1 ¿Por qué modelar?
7.1.2 Concepto de lenguajes de modelado
7.1.3 Modelos en el paradigma objetual
7.1.4 Necesidad de un estándar
7.1.5 ¿Qué es UML?
7.1.6 Lo que UML no es
7.1.7 Origen y evolución de UML

7.2 Una visión general de UML (30 Minutos)

7.2.1 Consideraciones generales
7.2.2 Las vistas
7.2.3 Los diagramas
7.2.4 Elementos de modelado
7.2.5 Mecanismos generales

7.3 Diagramas de estructura (2 Horas)

7.3.1 Definición
7.3.2 Diagramas de clase

7.4 Diagramas de interacción (1 Hora)
- 7.4.1 Definición
- 7.4.2 Diagramas de secuencia
- 7.4.3 Diagramas de colaboración

7.5 Casos de uso (2 Horas)
- 7.5.1 Introducción
- 7.5.2 Definición
- 7.5.3 Notación en UML
- 7.5.4 Relaciones entre casos de uso
- 7.5.5 Construcción de los casos de uso
- 7.5.6 Descripción de los casos de uso
- 7.5.7 Transición hacia los objetos

Tema 8. Visión general de la metodología OMT (1 Hora)

8.1 Introducción

8.2 Fases

8.3 Análisis
- 8.3.1 Definición del problema
- 8.3.2 Modelo de objetos
- 8.3.3 Modelo dinámico
- 8.3.4 Modelo funcional

8.4 Diseño del sistema

8.5 Diseño de los objetos

Unidad Docente IV: Miscelánea (2 Horas)

Tema 9. Herramientas CASE (2 Horas)

9.1 Introducción (30 Minutos)

9.2 Componentes de una herramienta CASE (10 Minutos)

9.3 Clasificación de las herramientas CASE (20 Minutos)

9.4 Integración de CASE (1 Hora)

Tabla 5.7. Estructura detallada del programa de teoría de Ingeniería del Software

Unidad Docente I: Conceptos Básicos

Objetivo genérico

Como su propio nombre indica, esta parte constituye una introducción necesaria para situar al alumno en el ambiente en el que se va a desarrollar su actividad.

Por otra parte, el discente debe tomar conciencia de que su trabajo profesional ha de ejecutarse con rigor, y por tanto, ha de estar sujeto al empleo de un método de ingeniería adaptado, en concreto, a la Ingeniería del Software.

Se pretende que el estudiante comprenda que está obligado a integrarse en proyectos interdisciplinares, con el objetivo y responsabilidad de sistematizarlos, para su coordinación en un trabajo, o conjunto de trabajos concretos que se realizarán con la asistencia de computadores. Así mismo, debe ser consciente que su labor implica la comunicación con diversas personas, cada una de las cuales jugará un rol determinado en el sistema de información en que se esté trabajando, siendo muchas veces necesario adaptarse a su visión del problema para que esa comunicación sea efectiva y redunde en una correcta solución del mismo.

El desarrollo de sistemas y aplicaciones, y la promoción en las organizaciones del uso masivo de las Tecnologías de la Información, suponen hoy un compromiso de eficiencia. Cada vez se exige una mayor interactividad entre las posiciones del organigrama en sus aspectos funcionales, para mejorar la cooperación y la integración de datos. Todo ello conlleva un aumento de la complejidad, que debe reducirse con el empleo de estándares de ingeniería que puedan ser considerados como patrones de diferentes niveles, y que aseguran el desarrollo de proyectos con la calidad, idoneidad, viabilidad y economía precisa.

Esta unidad supone el 13% de la asignatura, estando compuesta por un único tema, **Introducción a la Ingeniería del Software**, que se detalla a continuación.

Tema 1: ***Introducción a la Ingeniería del Software***

Descriptores

Software; aplicaciones del software; crisis del software; sistemas de información; tecnologías de información; ciclo de vida; paradigmas de la Ingeniería del Software; proceso software; metodología; método.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T1** y **T2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
- Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir a los alumnos en el planteamiento del ejercicio de la actividad de ingeniería.
- Establecer las características de los sistemas software.
- Establecer los conceptos de ciclo de vida del desarrollo del software, bajo los distintos planteamientos que se han ido desarrollando.
- Destacar la necesidad del empleo de metodología en el desarrollo del software, como única forma de desarrollar software profesional y de calidad.
- Establecer el desarrollo de software como un proceso interdisciplinar, que implica la colaboración del usuario, y que se ve favorecido por el conocimiento que tenga el ingeniero del software del dominio del problema.

Contenidos

1.1 Software
1.2 Sistemas de información
1.3 Ingeniería del Software
1.4 Ciclo de vida del software
1.5 Paradigmas de la Ingeniería del Software
1.6 Metodologías

Tabla 5.8. Contenidos del tema 1 del temario teórico de Ingeniería del Software

Resumen

En la primera parte se introducen los términos y conceptos que se manejan en el resto de temas, a la par que se comienza una reconducción del alumno, mediatizado por un culto excesivo a la tecnología, para que termine considerando que la tecnología no es un fin en sí misma, sino un medio para la construcción de los sistemas de información de las organizaciones.

En una segunda parte del tema, se estudia el ciclo de vida del software y del desarrollo del software, diferenciando, en una primera aproximación, las tres fases generales que aparecen en la construcción de todo producto software: *definición, desarrollo* y *mantenimiento*. Posteriormente, se estudian las características de los paradigmas de la Ingeniería del Software (*también denominados modelos de ciclo de vida*) más representativos.

La exposición que de los diferentes modelos de ciclo de vida se hace, conduce al alumno desde los modelos secuenciales, totalmente desaconsejables, hasta los modelos evolutivos e incrementales, más acordes con las herramientas de desarrollo actuales y con la Orientación a Objetos. Como final de esta parte se presenta *el estándar ISO 12207* [ISO/IEC, 1995], como el marco estándar donde encuadrar los diferentes procesos que aglutinan las actividades del ciclo de vida del software, y una breve reseña a los *modelos de ciclo de vida en los desarrollos orientados a objetos*, presentado como ejemplo concreto el modelo fuente [Henderson-Sellers and Edwards, 1990].

La tercera y última parte del tema está dedicada a la introducción de las metodologías de desarrollo, como contraposición al desarrollo anárquico y artesanal de aplicaciones, tan relacionado con la tan nombrada *crisis del software*.

Al clasificar las metodologías se hace una comparativa entre las características de aquéllas centradas en el paradigma estructurado frente a las metodologías orientadas al objeto.

Bibliografía

- ***Citada en las transparencias del tema:***

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[Andreu et al., 1996] Andreu, Rafael, Ricart, Joan y Valor Josep. "*Estrategia y Sistemas de Información*". 2ª Ed. McGraw-Hill (*serie de management*), 1996.

[Blum, 1992] Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, p. 20, 1992.

[Boehm, 1986] Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". ACM Software Engineering Notes, 11(4):22-42. 1986.

[Boehm, 1988] Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

[Buxton et al., 1976] Buxton, J. M., Naur, P. and Randell, B. (eds.) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[CERN, 1997] CERN. "*STING Software Engineering Glossary*". CERN. http://dxsting.cern.ch/sting/glossary.html. April 1997.

[Coleman et al., 1994] Coleman, D., Arnold, P. and Bodoff, S. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

[DoD, 1995] DOD. "*SRI Reuse Glossary*". DOD, http://sw-eng.falls-church.va.us/ReuseIC/policy/glossary/glossary.htm, December 1995.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Emery, 1990] Emery, J. C. "*Sistemas de Información para la Dirección. El Recurso Estratégico y Crítico*". Ed. Díaz de Santos, 1990.

[Frakes et al., 1991] Frakes, William B., Fox, Christopher, Nejmeh and Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

[Henderson-Sellers and Edwards, 1990] Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

[Horan, 1995] Horan, Peter "*Software Engineering - A Field Guide***"**. Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[Houghton, 1992] Houghton Mifflin Company. "*The American Heritage Dictionary of the English Language*". 3rd Edition, Houghton Mifflin Company, Electronic Version. 1992.

[Humphrey, 1993] Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science,* Van Nostrand Reinhold, p. 1218. 1993.

[IEEE, 1999a] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[IEEE, 1999b] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 2: Process Standards*". IEEE Computer Society Press, 1999.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Leany, 1995] Leaney, John. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. http://www.ee.uts.edu.au/~jrleaney/setut.htm. October 1995.

[Maddison, 1983] Maddison, R. N. "*Information System Methodologies*". Wiley Henden, 1983.

[Monforte, 1995] Monforte, Manfredo. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

[NIST, 1994] National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

[Parnas and Weiss, 1987] Parnas, David Lorge and Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4):259-265, December 1987.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Royce, 1970] Royce, W. W. "*Managing the Development of Large Software Systems: Concepts and Techniques*". In Proceedings WESCON. August, 1970.

[Rubin and Goldberg, 1992] Rubin, Kenneth S. and Goldberg, Adele. "*Object Behavior Analysis*". Communications of the ACM 35(9): 48-62. September, 1992.

[Rumbaugh, 1995] Rumbaugh, James. "*What Is a Method*". JOOP. October, 1995.

[Rumbaugh et al. 1991] Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Shaler and Mellor, 1992] Shaler, S. and Mellor, S. "*Object Life Cycles: Modeling the World in States*". Prentice-Hall, 1992.

[Wirfs-Brock et al., 1990] Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wolff, 1989] Wolff, J. G. "*The Manegement of Risk System Development: 'Project SP' and the 'New Spiral Model'*". Software Engineering Journal, May 1989.

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

- ***Lecturas complementarias:***

"*Anecdotes/stories about Software Engineering*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/anecdote. [Última vez visitado, 28-1-2000]. July, 1993.

Anécdotas e historias sobre Ingeniería del Software.

"*Computer Horror Stories*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/horror. [Última vez visitado, 28-1-2000]. July, 1993.

Historias macabras de fallos de ordenadores.

"*Origin of Term 'Software Engineering'*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/SEorigin. [Última vez visitado, 28-1-2000]. July, 1993.

Curiosidades sobre el origen del nombre de Ingeniería del Software.

"*Software Engineering Questions and Answers*". http://www.qucis.queensu.ca/FAQs/comp.software-eng/questions.html. [Última vez visitado, 28-1-2000]. 1998.

Versión hipertexto de la respuestas breves a las preguntas más frecuentes en el grupo de noticias **comp.software-eng**.

Baber, Robert L. "*Comparison of Electrical 'Engineering' of Heaviside's Times and Software 'Engineering' of our Times*". IEEE Annals of the History of Computing, 19(4):5-17. 1997.

Artículo que realiza una comparación de los problemas actuales de la Ingeniería del Software, con los problemas que tuvieron otras disciplinas que hoy en día se consideran ingenierías en sus principios.

Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

En este artículo Barry Boehm expone su modelo en espiral y su utilización en **TRW**, lo que permite entender cómo se van definiendo las distintas vueltas de la espiral.

Boehm, Barry W., Egyed, Alexander, Kwan, Julie, Port, Dan, Shah, Archita and Madachy, Ray. "*Using the WinWin Spiral Model: A Case Study*". IEEE Computer, 31(7):33-44, July, 1998.

En este artículo se presenta la aplicación práctica del modelo de ciclo de vida en espiral WinWin, una extensión del ciclo de vida definido por Boehm, al que se le ha añadido las actividades de la *Teoría W* al comienzo de cada ciclo.

Brereton, Pearl, Budgen, David, Bennet, Keith, Munro, Malcom, Layzell, Paul, Macaulay, Linda, Griffiths, David and Stannett, Charles. "*The Future of Software*". Communications of the ACM, 42(12):78-84. December, 1999.

Artículo que identifica los puntos clave (*accesibilidad, adapatabilidad, transparencia, seguridad y manejabilidad*) en los que trabajar en aquellas organizaciones cuyo negocio depende directamente del rendimiento de sus sistemas software.

Chandra, Jagdish, March, Salvatore T., Mukherjee, Satyen, Pape, Will, Ramesh, R., Rao, H. Raghav and Waddoups, Ray O. "*Information Systems Frontiers*". Communications of the ACM, 43(1):71-79. January, 2000.

Artículo que trata sobre el crecimiento y aplicación de los sistemas de información, junto a las Tecnologías de la Información, en dominios de aplicación que nunca habían sido considerados.

Fayad, Mohamed E., Laitinen, Mauri and Ward, Robert P. "*Software Engineering in the Small*". Communications of the ACM, 43(3):115-118. March, 2000.

Artículo donde se defiende que las compañías que desarrollan proyectos software de pequeño tamaño también deben utilizar técnicas de Ingeniería del Software.

Glass, Robert L. "*The Software Criris… Not?*". IEEE Computer, 27(4):104. April, 1994.

Opinión personal del autor del artículo sobre la crisis del software.

Gutiérrez, Inmaculada y Medinilla, Nelson. "*Contra el Arraigo de la Cascada*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. Pere Botella, Juan Hernández y Félix Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999.

Trabajo crítico con el modelo de ciclo de vida en cascada, realizado desde la perspectiva de la complejidad de la incertidumbre en los proyectos software.

Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

En este artículo se describe el modelo de ciclo de vida fuente para desarrollos orientados a objetos.

Jacobson, Ivar. "*Building Without Blueprints*". Object Magazine, 7(9): 71-72. November, 1997.

Artículo que resalta la importancia de los modelos software.

Leaney, John. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. http://www.ee.uts.edu.au/~jrleaney/setut.htm. [Última vez visitado, 28-1-2000]. Octubre 1995.

Conjunto de páginas web que dan un somero repaso a diferentes aspectos de la Ingeniería del Software.

Lions, J. L. "*ARIANE 5 Flight 501 Failure*". Report by the Inquiry Board. http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html. [Última vez visitado, 28-1-2000]. París, 19 Julio 1996.

Informe de las causas del fallo del lanzamiento de la lanzadera espacial Ariane 5 el 4 de Junio de 1996.

Raccoon, L. B. S. "*Fifty Years of Progress in Software Engineering*". Software Engineering Notes (SEN), 22(1):88-104. January, 1997.

Presenta una visión de la evolución de la Ingeniería del Software. Presenta también el modelo de ciclo de vida **Caos**, comparándolo con otros modelos.

Singh, Raghu. "*The Software Life Cycle Processes Standard*". IEEE Computer, 28(11):89-90. November, 1995.

Visión esquemática del estándar ISO/IEC 12207 sobre los procesos que componen el ciclo de vida del software.

- ***Referencias utilizadas para preparar las clases:***

Amescua Seco, Antonio de, García Sánchez, Luis, Martínez Fernández, Paloma y Díaz Pérez, Paloma. "*Ingeniería del Software de Gestión. Análisis y Diseño de Aplicaciones*". Paraninfo, 1995.

Libro que se centra en la aplicación de una metodología estructurada para el desarrollo del software. En concreto toma como referencia SSADM en su versión 4.

Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

Definiciones de términos relacionados con la Ingeniería del Software.

Boehm, Barry W. "*Software Engineering Economics*". Prentice Hall, 1981.

En su cuarto capítulo, **The software life-cycle: phases and activities**, se centra en el modelo en cascada, presentando diferentes variantes.

Boehm, Barry W. and Bose, Prasanta. "*A Collaborative Spiral Software Process Model Based on Theory W*". In Proceedings of the Int'l Conf. Software Process. IEEE Computer Society Press. Pages, 59-68. 1994.

Este artículo presenta una extensión del modelo en espiral llamado *modelo de la siguiente generación* (*Next Generation Process Model - NGPM*).

Conger, Sue. "*The New Software Engineering*". Course Technology, 1994.

Libro general de Ingeniería del Software cuyo primer capítulo, **Overview of Software Engineering**, se corresponde con los objetivos buscados en este primer tema: *presentación de conceptos, introducción al ciclo de vida* y *descripción de los tipos de metodologías*.

Dedene, G. and Snoeck. "*MERODE: A Model-Driven Entity-Relationship Object-Oriented Development Method*". ACM Software Engineering Notes, 19(3):51-61. July, 1994.

Artículo que describe un método orientado a objetos de primera generación, descendiente de las técnicas estructuradas. Incluye una comparativa de los métodos de desarrollo orientados a objetos coetáneos a éste.

Frakes, William B., Fox, Christopher J. and Nejmeh, Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

En su primer capítulo de introducción se pueden encontrar algunas anécdotas ocurridas en proyectos software, mientras que en su segundo capítulo, **Concept exploration and requirements specification**, se aboga por el prototipado como ayuda en las primeras fases del ciclo de vida.

Gaitero Gordillo, Domingo. "*Metodología Métrica. Un Enfoque Práctico*". Everest Multimedia, 1997.

Libro que trata sobre la metodología métrica. Se puede utilizar como complemento a [MAP, 1995].

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto para el desarrollo de este tema se adeúan: el tema 1, *Introducción a la Ingeniería del Software*, que sirve como presentación a la problemática de la realización de proyectos software y del vocabulario que se maneja en el resto de los temas; y el tema 2, *Ciclo de Vida del Proyecto*, donde se presentan algunos de los modelos de ciclo de vida más tradicionales.

Ghezzi, Carlo, Jazayeri, Mehdi and Mandrioli, Dino. "*Fundamentals of Software Engineering*". Prentice-Hall International, 1991.

Libro general de Ingeniería del Software del que cabe destacar en relación con el presente tema su capítulo 7, **The software production process**, en el que se presentan los modelos de ciclo de vida en cascada, evolutivo, de transformación y en espiral, donde merece una mención especial el desarrollo que hace de este último.

Gray, Lewis. "*ISO/IEC 12207 Software Lifecycle Processes*". Crosstalk. The Journal of Defense Software Engineering, 9(8). August, 1996.

Artículo que describe la norma ISO/IEC 12207, comparándolo con el estándar militar de EEUU MIL-STD-498.

Henderson-Sellers, Brian. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

En su capítulo 4, **Object-Oriented Systems Development**, trata el tema del ciclo de vida comparando el ciclo de vida clásico con el ciclo de vida que siguen los sistemas orientados a objetos, presentando también su modelo fuente. En este mismo capítulo hace una clasificación de las metodologías en: *metodologías de descomposición funcional, metodologías orientadas a objetos puristas, metodologías híbridas* y *metodologías convergentes*.

Kruchten, Philippe. "*A Rational Development Process*". Crosstalk, 9(7):11-16. July, 1996.

En este artículo el Dr. Kruchten, director de Proceso de Desarrollo en Rational Software Corporation, expone las bases de un proceso de desarrollo software evolutivo e incremental, que posteriormente se convertiría en el RUP (Rational Unified Process).

Lucero, José Luis. "*Gestión de la Calidad del Software*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Abril 1996.

En el primer capítulo de la documentación de este curso se encuentra una amplia referencia al ciclo de vida del software así como a varios paradigmas concretos (páginas 1.16 - 1.37).

Lucero, José Luis y Ramos, Miguel Ángel. "*Dirección de Departamentos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Febrero-Marzo 1996.

En la documentación de este curso se puede encontrar una visión generalista del ciclo de vida de un desarrollo (páginas V.7-V.13).

Lucero, José Luis y Ramos, Miguel Ángel. "*Gestión de Proyectos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Enero-Febrero 1996.

El segundo capítulo de la documentación de este curso esta dedicado en su mayor parte al ciclo de vida de los sistemas de información (páginas 2.1 - 2.34). Tras una breve introducción del concepto de ciclo de vida, describe algunos de los principales modelos (cascada, evolutivo, transformación, prototipo y espiral) destacando especialmente el enfoque del modelo en cascada y del modelo en espiral con el plan de gestión de riesgos.

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Guía de referencia de la metodología Métrica 2.1.

Moore, Jim. "*ISO 12207 and Related Software Life-Cycle Standards*". http://www.acm.org/tcs/lifecycle.html. [Última vez visitado, 12-3-2000]. 1996.

Introducción al modelo de ciclo de vida ISO 12207.

Muller, Pierre-Alain. "*Modelado de Objetos con UML*". Ediciones Gestión 2000, 1997.

Este libro, aunque dedicado a UML, en su capítulo 4, **Encuadre de los proyectos orientados a objetos**, hace una descripción muy interesante del ciclo de vida iterativo e incremental, comparándolo con el ciclo de vida lineal y presentado variantes del ciclo iterativo.

Pfleeger, Shari Lawrence. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Relacionados con este primer tema se recomienda la consulta de los dos primeros capítulos del libro. El capítulo 1, **Why software engineering**, hace una introducción muy interesante a la Ingeniería del Software, justificando su existencia. Por su parte el capítulo 2, **Modeling the process and life cycle**, describe el proceso de creación de un sistema software, el ciclo de vida del software, describiendo varios modelos de ciclo vida: *modelo en cascada, el modelo en V, modelo de prototipos, modelo de transformación, modelo incremental e iterativo* y *modelo en espiral*.

Piattini Velthuis, Mario Gerardo. "*Ciclos de vida para Sistemas Orientados a Objetos*". Cuore, (7):6-11. Septiembre, 1995.

Análisis comparativo de algunos modelos de ciclo de vida propuestos para el desarrollo de sistemas software orientados a objetos.

Piattini, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

Cabe destacar el *capítulo 3: Ciclo de Vida del Software* (páginas 39-60) en el que, además de los conceptos genéricos sobre el ciclo de vida, presenta en mayor detalle el modelo en cascada, el modelo incremental y el modelo en espiral. Este capítulo incluye una interesante referencia a diversos paradigmas para desarrollos orientados a objeto (*modelo de agrupamiento, modelo fuente,*

modelo remolino y modelo pinball). Además, hace un resumen muy adecuado del estándar ISO/IEC 12207.

También es interesante el *capítulo 4: Metodologías de Desarrollo de Software* (páginas 61-88). En este capítulo, además de la presentación de unos conceptos generales, diferencia los conceptos de metodología, ciclo de vida y método, incluyendo también un somero repaso de las metodologías estructuradas más difundidas en la actualidad.

Pressman, Roger S. "*Ingeniería del Software: Un Enfoque Práctico*". 3ª Edición, McGraw Hill, 1993.

Traducción de [Pressman, 1992]. Por lo que se refiere a los ciclos de vida, en su primer capítulo están los apartados 1.5 y 1.6 (páginas 24-38) en los que, además de una visión general del ciclo de vida, se detalla con mayor profundidad el modelo clásico, el modelo de prototipos, el modelo en espiral y las técnicas de cuarta generación.

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición, McGraw Hill, 1998.

Traducción al español de [Pressman 1997]. En la última de edición hasta la fecha de este clásico, Roger S. Pressman cambia la forma de abordar el tema del ciclo de vida del software. En su capítulo 2 dedicado al *proceso software*, presenta el modelo lineal secuencial (*modelo clásico o en cascada*), el modelo de prototipos, el modelo DRA (*Desarrollo Rápido de Aplicaciones*), los modelos de procesos evolutivos de software (*dentro de los que estudia el modelo incremental, el modelo en espiral, el modelo de ensamblaje de componentes y el modelo de desarrollo concurrente*), el modelo de métodos formales y las técnicas de cuarta generación.

Scacchi, Wait. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). October, 1987.

Presentación del concepto de ciclo de vida desde su origen histórico y su necesidad. Describe algunos de los ciclos de vida más tradicionales (clásico, espiral, estándares militares...) y comenta algunas de las referencias bibliográficas que hoy se consideran como clásicas.

Sommerville, Ian. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

Esta es una referencia clásica dentro de la Ingeniería del Software. En relación con el tema presenta en su primer capítulo, **Introduction**, una breve introducción a la Ingeniería del Software, en la que, someramente, presenta el ciclo de vida en cascada y en espiral.

Telefónica. "*MARTE: Metodología Armonizada de Telefónica*". Telefónica. 1998.

Documentación interna de la metodología seguida por Telefónica DGOSI, Telefónica I+D, Telefónica Móviles y Telefónica Sistemas para el desarrollo y gestión de proyectos software.

TOA. "*A Comparison of Object-Oriented Methodologies*". The Object Agency, Inc. 1995.

Completo informe comparativo de diversos métodos de primera generación.

Tomayko, James E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

Artículo que echa un vistazo a la historia de la Ingeniería para encontrar ejemplos consistentes con el estado de la práctica actual de la Ingeniería del Software.

Yourdon, Edward. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

El capítulo 5, **El ciclo de vida del proyecto**, donde, entre otros modelos, se presenta el ciclo de vida estructurado.

Unidad Docente II: Paradigma Estructurado de Desarrollo

Objetivo genérico

Aunque la incorporación, cada vez mayor, a las empresas de nuevos titulados en Informática está influyendo de forma decisiva para que éstas vayan adoptando poco a poco los nuevos avances tecnológicos, este proceso es lento y conservador.

En un contexto donde un gran número de empresas y organizaciones no utilizan ningún tipo de metodología, y de las que la utilizan, una gran parte de ellas emplean alguna basada en los planteamientos del paradigma estructurado, los nuevos titulados deben estar preparados para incorporarse a los proyectos que se desarrollen bajo este paradigma, mantener el enorme parque de aplicaciones legadas y estar en disposición de participar en la migración metodológica que, hacia la Orientación a Objetos, van llevando a cabo cada vez más empresas.

Además, los currículos de las Ingenierías Técnicas en Informática de la Universidad española son, en general, muy conservadores; tomando como centro de referencia este paradigma desde que el alumno cursa sus primeras asignaturas propias de Informática.

Así pues, los objetivos genéricos de esta unidad docente son *conocer la evolución del análisis y el diseño en el paradigma estructurado* así como *conocer y manejar las técnicas del análisis y diseño estructurado orientado a procesos y a datos*.

Esta unidad docente ocupa el 52% del temario de teoría de la asignatura, estando compuesta por cuatro temas (**Análisis y especificación de requisitos**, **Análisis estructurado, Diseño del software** y **Diseño estructurado**), que se detallan a continuación.

Tema 2: ***Análisis y Especificación de Requisitos***

Descriptores

Análisis del sistema, análisis de requisitos, requisito, obtención (elicitación) de requisitos, especificación de requisitos, ERS, modelo, modelado funcional, modelado de información, modelado del comportamiento, DFD, descomposición en procesos, flujos de datos, entidades externas, almacenes de datos, extensiones de los DFD para sistemas en tiempo real, diccionario de datos, miniespecificación, diagrama entidad-relación, diagrama de transición de estados, balanceo de modelos.

Objetivos

Este tema se orienta a satisfacer los objetivos **T3** y **T4** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en la problemática de la obtención, gestión, análisis, documentación y especificación de los requisitos de un sistema software.
- Presentar los diferentes tipos y las distintas vistas de los requisitos, en concreto distinguir entre los requisitos del cliente (*Requisitos-C*) y los requisitos del desarrollador (*Requisitos-D*) [Brackett, 1990].
- Capacitar al estudiante para comprender y realizar un análisis utilizando técnicas estructuradas en las que se introduce en sus aspectos teóricos y prácticos más relevantes.
- Lograr que el alumno entienda las ventajas de utilizar técnicas de especificación gráficas, que faciliten la obtención de modelos y favorezcan la comunicación con clientes y/o usuarios.

Contenidos

2.1 Introducción al análisis
2.2 Especificación de requisitos del software
2.3 Técnicas de especificación
2.4 Modelado funcional
2.5 Modelado de la información
2.6 Modelado del comportamiento
2.7 Balanceo de modelos

Tabla 5.9. Contenidos del tema 2 del temario teórico de Ingeniería del Software

Resumen

El tema se ha dividido en siete apartados principales. El primero de ellos sirve para realizar una presentación de lo qué se entiende por análisis, sirviendo de enlace con lo introducido en el tema anterior al hablar del ciclo de vida del software.

Antes de entrar a definir qué se entiende por análisis y por requisito, debe diferenciarse lo que se entiende por análisis del sistema y análisis de requisitos, núcleo del presente tema, así como introducir lo que se entiende por Ingeniería de Requisitos (*compuesta por el propio análisis de requisitos, la especificación de los mismos y el prototipado*).

Completa esta introducción un repaso a las tareas y principios fundamentales de esta fase del ciclo de vida del software.

El segundo apartado de este tema se centra en el documento por excelencia de esta fase, la *Especificación de Requisitos del Software (ERS)*, que englobaría los requisitos del cliente como los requisitos del desarrollador, aunque hay corrientes metodológicas que abogan por la división de la ERS en dos documentos, el primero de ellos se centraría en los requisitos del cliente, denominándose *Documento de Requisitos del Sistema – DRS* [Durán y Bernárdez, 1999b], mientras que el segundo recogería los requisitos del desarrollador, denominándose *Documento de Análisis del Sistema - DAS* [Durán y Bernárdez, 1999a]. La metodología Métrica 2.1 [MAP, 1995] es otro ejemplo en el que se tienen dos documentos separados para los requisitos-C y los requisitos-D.

Con el concepto de ERS, la exposición de sus características y la presentación de su estructura según algunos estándares, como el IEEE Std 830-1998, *Recommended Practice for Software Requirements Specifications* [IEEE, 1999], o según alguna metodología [Durán y Bernárdez, 1999a], [Durán y Bernárdez, 1999b], se busca que el alumno se acostumbre a documentar los elementos software que realiza, independientemente del nivel de abstracción de éstos, y a que esta documentación se ajuste a unos estándares, ya sean internacionales o impuestos por una metodología concreta.

El tercer apartado se dedica a clasificar las diferentes técnicas de especificación según dos criterios diferentes: *su modo de representación* (gráficas, textuales, marcos y matriciales) y *su enfoque de modelado* (funcional, datos y comportamiento), donde este último será el que se siga en el resto del tema para presentar las técnicas más extendidas.

En el cuarto apartado se presentan la técnica más representativa del modelado funcional, y del análisis estructurado orientado a procesos, el *Diagrama de Flujo de Datos – DFD*, y de las técnicas accesorias que completan la información que en él aparece: el *Diccionario de Datos* y la *Especificación de Procesos*.

Aunque se presentan las notaciones de estos diagramas según Yourdon [Yourdon, 1989], Tom De Marco [DeMarco, 1979], Gane y Sarson [Gane and Sarson, 1981] y Métrica 2.1 [MAP, 1995], la notación seguida es la de Yourdon.

Se establecen los conceptos fundamentales de los DFDs, tales como: *entidades externas, flujos de datos, procesos* y *almacenes*.

En este tema se define como planteamiento básico, para esta perspectiva, el reiterado principio de descomposición por explosión de un proceso en otros más elementales, hasta que no se considere necesario un mayor detalle, lo que constituye la esencia del método.

Se introducen las normas o pautas que deben seguirse para la confección de diagramas de flujo de datos, citándose los errores más frecuentes.

Se menciona expresamente la necesidad de que los sistemas estén *balanceados*, lo que significa que en la explosión de un proceso no deben aparecer nuevas entidades externas ya que éstas, precisamente por su carácter, no deben estar ocultas dentro de ningún proceso. Por el contrario si es posible que aparezcan, por ser elementos internos del proceso que se está descomponiendo, almacenes de datos.

Se destaca que cuando los DFDs se utilizan en las reuniones del analista con los usuarios, no se deben profundizar exageradamente en el detalle, ya que se convertirían en farragosos, si no inútiles, para esta finalidad. Por el contrario, cuando se utilizan para comunicarse con desarrolladores es pertinente utilizar diagramas más detallados.

Para la especificación de sistemas en tiempo real se presentan las extensiones a la notación de Yourdon realizadas por Ward y Mellor [Ward and Mellor, 1985] y por Hatley y Pirbhai [Hatley and Pirbhai, 1987].

Se define el concepto de *diccionario de datos* como una lista organizada de todos los conceptos que son significativos para el sistema con sus definiciones, por lo tanto se describirán y se definirán: *los flujos de datos, tanto de entrada como de salida, los almacenes de datos, y los elementos correspondientes tanto a los flujos de datos como a los almacenes*.

Aunque es cierto que los requisitos recibidos de los usuarios deben expresar las necesidades de información, lo que conduce a analizar el *qué se debe hacer*, dejando para la fase de diseño e implementación el *cómo hacerlo*. Sin embargo, para que la especificación del modelo funcional quede completa se debe contar con las técnicas oportunas para la especificación de la lógica de los procesos primitivos, que eviten las ambigüedades que se derivan del uso de descripciones literales.

Entre las técnicas a utilizar para la *especificación de los procesos primitivos* se puede citar entre otras: *lenguaje estructurado, pre y postcondiciones, tablas de decisión, árboles de decisión* y *diagramas de acción*.

Es bien conocido que estas herramientas tienen también utilidad en la fase de diseño, cuando se trata de explicar a los programadores el tipo de algoritmos que se deben emplear. Pero no es ese el sentido que aquí se le da, ya que de nuevo se trata de aplicarlas para lograr un diálogo, claro y conciso con el usuario.

En el apartado cinco se dedica al modelado de la información, haciendo hincapié que dentro del paradigma estructurado la técnica por excelencia es el diagrama entidad-interrelación [Chen, 1976], pero no se explica esta técnica porque se considera que fue objeto de su estudio en la asignatura de 2º curso **Diseño de Bases de Datos**, aunque se reitera su importancia y su utilización en la presente asignatura.

El sexto apartado está dedicado al modelado de comportamiento, donde, y de nuevo por la falta de tiempo, se estudian los *Diagramas de Transición de Estados – DTE*, como la técnica más utilizada en los sistemas donde el tiempo es un factor crítico. Esta técnica sirve para especificar los estados globales de un sistema o como técnica de especificación de un proceso de control.

Otra técnica de suma importancia son las *Redes de Petri* para la descripción en sistemas de comportamiento asíncrono y concurrente. Este formalismo queda fuera de temario por las restricciones temporales, pero se aconseja su estudio mediante alguna de las lecturas complementarias recomendadas en este tema, o bien mediante la realización de algún seminario.

Por último, el séptimo apartado establece, a forma de recetario, una serie de recomendaciones para controlar la eliminación de incongruencias entre los tres modelos fundamentales del sistema (*funcional, datos* y *comportamiento*), siendo una aportación interesante la utilización de técnicas matriciales.

Bibliografía

- ***Citada en las transparencias del tema:***

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Durán y Bernárdez, 1998] Durán, A. y Bernárdez, B. "*Norma para la Recolección de Requisitos de un Sistema Software (versión 1.1)*". Apéndice en las Actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. Murcia, Noviembre 1998. También disponible en línea en http://www.lsi.us.es/~amador/norma/recoleccion.zip. [Última vez visitado: 12/1/2000] y en http://tejo.usal.es/~fgarcia/docencia/isoftware/doc/norma.pdf [Última vez visitado: 12/1/2000]. 1998.

[Hatley and Pirbhai, 1987] Hatley, Derek J. and Pirbhai, Imtiaz. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

[IEEE, 1999a] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[IEEE, 1999b] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 2: Process Standards*". IEEE Computer Society Press, 1999.

[IEEE, 1999c] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 4: Resource and Technique Standards*". IEEE Computer Society Press, 1999.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pressman, 1992] Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Raghavan et al., 1994] Raghavan, S., Zelesnik, G. and Ford, G. "*Lecture Notes on Requirements Elicitation*" CMU/SEI-94-EM-10, Pittsburgh (EEUU), Software Engineering Institute (Carnegie Mellon University). 1994.

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Ward and Mellor, 1985] Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

- ***Lecturas complementarias:***

Boehm, Barry and Port, Dan. "*When Models Collide: Lessons from Software Systems Analysis*". IEEE IT Professional, 1(1):49-56. January/February, 1999.

> Artículo que expone cómo reconocer las presunciones que, sobre el sistema software a construir, tienen las diferentes personas que participan en su construcción, y reaccionar en consecuencia.

Chance, Brian D. and Melhart, Bonnie E. "*How to Develop Better System Requirements*". IEEE IT Professional, 1(3):70-72. May/June, 1999.

> En este artículo se establecen cinco tipos de escenarios (*operacionales, de refinamiento, de fallo, de rendimiento* y *de aprendizaje*) para completar los requisitos de un sistema software.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 18 de octubre de 1999.

> Metodología para la recolección de requisitos de un sistema software. Define como único producto entregable el Documento de Requisitos del Sistema, que documenta los requisitos-C utilizando una combinación de casos de uso, plantillas para la documentación de requisitos y patrones lingüísticos propios del dominio de aplicación.

Gardarin, Georges. "*Dominar las Bases de Datos*". Ediciones Gestión 2000, 1993.

> Un buen libro sobre bases de datos en el que se trata el diagrama entidad-relación.

Silva, Manuel. "*Las Redes de Petri: En la Automática y la Informática*". Editorial AC, libros científicos y técnicos. Madrid. 1985.

Para profundizar en las redes de Petri.

- ***Referencias utilizadas para preparar las clases:***

Abernethy, Ken and Kelly, John C. "*Comparing Object-Oriented and Data Flow Models – A Case Study*". In Proceedings of the 1992 ACM Computer Science 20th annual conference on Communications, CSC '92. (March 3-5, 1992, Kansas City, MO - USA). Pages 541-547. ACM. 1992.

Informe que compara las técnicas estructuradas, en concreto los DFDs, con los modelos realizados en la Orientación a Objetos.

Battini, C., Ceri, S. and Navathe, S. B. "*Conceptual Database Design: An Entity Relationship Approach*". Benjamin/Cummings, 1992.

Se trata de un libro que, fiel a su título, está por entero dedicado al modelado conceptual de bases de datos a través de la aproximación del modelo entidad-interrelación. Aborda aspectos metodológicos y de análisis fundamentalmente; pero también entra en temas de prácticos del diseño lógico.

Es por tanto un libro que se encuentra a medio camino entre la Ingeniería del Software y las Bases de Datos. Constituye un texto complementario ideal, con interesantes propuestas de ejercicios y estudios de casos prácticos.

Este texto se encuentra traducido al castellano: "*Diseño de Bases de Datos: Un Enfoque de Entidades-Interrelaciones*". Addison-Wesley/Díaz de Santos, 1994.

Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1990.

Información útil para el tratamiento del tema de los requisitos software en la docencia. Referencias comentadas de artículos clásicos. Distinción entre requisitos de cliente (*requisitos-C*) y requisitos del desarrollador (*requisitos-D*).

Burns, Thomas, Fong Elizabeth, Jefferson, David, Knox, Richard, Mark, Leo, Reedy, Christopher, Reich, Louis, Roussopoulos, Nick and Truszkowski, Walter. "*Reference Model for DBMS Standardization*". SIGMOD RECORD, 15(1). March, 1986.

Informe que establece los diferentes niveles en las Bases de Datos Relacionales.

Conger, Sue. "*The New Software Engineering*". Course Technology, 1994.

El capítulo 7, **Process oriented analysis**, de este libro puede servir para la obtención de algún ejemplo para desarrollar en clase o para realizar en prácticas.

Champeaux, Dennis de (moderator), Constantine, Larry, Jacobson, Ivar, Mellor, Stephen, Ward, Paul and Yourdon, Edward. "*PANEL: Structured Analysis and*

Object Oriented Analysis". In Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA90/ECOOP90. (October 21 - 25, 1990, Ottawa Canada). Pages 135-139. ACM, 1990.

Opinión, en 1990, de algunos de los mayores expertos sobre las técnicas de análisis estructuradas y orientadas al objeto.

Christel, Michael G. and Kang, Kyo C. "*Issues in Requirements Elicitation*". Technical Report CMU/SEI-92-TR-12 (ESC-TR-92-012). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). September, 1992.

En este informe se hace una introducción al tema de la obtención de requisitos, presentando el proceso asociado, las técnicas que se utilizan y los problemas que se tienen en esta etapa del desarrollo del software. Además, se plantea un marco metodológico para su puesta en práctica.

Davis, Alan M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

Libro centrado en las primeras fases del ciclo de vida, en concreto en las técnicas de especificación de los requisitos. Recoge un buen número de técnicas para la especificación de modelos funcionales, de datos y de comportamiento. Las técnicas orientadas a objetos están presentes aunque no es la mejor referencia para su estudio. Incluye diferentes estándares para la organización de un documento de especificación de los requisitos del software.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel y Ruiz Cortés. "*Una Propuesta Metodológica para la Recolección de Requisitos de un Sistema Software*". En las Actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. (Murcia, 13 y 14 de Noviembre 1998). Páginas 37-48. 1998.

Resumen de la metodología para la recolección de requisitos de un sistema software en su versión 1.1, desarrollada por el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel and Ruiz Cortés. "*An Object-Oriented Model and a CASE Tool for Software Requirements Management and Documentation*". In Proceedings of the 4th Workshop MENHIR. Francisco José García and José Manuel Marqués editors. (May 6-7, 1999, Sedano, Burgos – Spain). Pages 6-10. 1999.

Presenta un modelo orientado a objetos de los requisitos software.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel and Ruiz Cortés. "*Elicitación de Requisitos de Usuario Mediante Plantillas y Patrones de Requisitos*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. Pere Botella, Juan Hernández y Félix Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 183-194. 1999.

Utilización de patrones lingüísticos en la documentación de requisitos. Distinguen entre patrones-R (plantillas ya rellenas) y patrones-L (frases con huecos). Presenta un modelo de procesos para la Ingeniería de Requisitos.

Ellison, Karen S. "*Developing Real-Time Embedded Software in a Market-Driven Company*". John Wiley & Sons, 1994.

Libro que aborda el desarrollo de sistemas software de tiempo real.

Gaitero Gordillo, Domingo. "*Metodología Métrica. Un Enfoque Práctico*". Everest Multimedia, 1997.

Una fuente donde encontrar algún ejemplo semidesarrollado para facilitárselo a los alumnos como uso práctico de las técnicas o incluso como material para las prácticas.

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto el tema 4, *Análisis de Requisitos del Software*, que se centra en las técnicas estructuradas de análisis, tanto para el modelado funcional como para el modelado de información y de comportamiento.

Hansen, Gary W. and Hansen, James V. "*Diseño y Administración de Bases de Datos*". 2ª Edición. Prentice Hall, 1997.

Libro de bases de datos (*traducido de Database Management and Design, Second Edition. Prentice Hall, 1996*) donde se dedica el capítulo 4, **Principios del diseño conceptual de bases de datos**, al modelo de información, utilizando un modelo entidad-relación extendido, muy semejante en el fondo, a los diagramas de clase de la Orientación a Objetos. Hay algunos ejemplos que pueden ser aprovechados para prácticas, ejemplos en clase o exámenes.

Hatley, Derek J. and Pirbhai, Imtiaz. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

Libro que se centra en el desarrollo de sistemas software de tiempo real. Estos autores definen una extensión a la notación de Yourdon para la especificación de estos sistemas.

Hawryszkiewycz, I. T. "*Introducción al Análisis y Diseño de Sistemas con Ejemplos Prácticos*". Anaya Multimedia, 1990.

Libro que no aporta nada en cuanto a las técnicas, siendo lo más destacable los ejemplos que incluye, que pueden aprovecharse para elaborar supuestos prácticos y exámenes.

Hofmann, Hubert F. "*Requirements Engineering*". Technical Report 93.05. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. March, 1993.

Informe técnico que se centra en la importancia de los requisitos en el ciclo de vida del software. Presenta un pequeño estado del arte de las técnicas y los métodos para la obtención de requisitos.

Jones, Trevor H., Song, Il-Yeol and Park, E. K. "*Ternary Relationship Decomposition and Higher Normal Form Structures Derived from Entity Relationship Conceptual Modeling*". In Proceedings on 1996 ACM on Computer science conference, CSC '96. (Feb. 16-18, 1996, Philadelphia, PA - USA). Pages 96-104. ACM. 1996.

Artículo que aborda la utilización de las relaciones ternarias en el modelado conceptual y su posterior paso al diseño.

Kowal, James A. "*Behavior Models. Specifying User's Expectations*". Prentice-Hall International, 1992.

Presenta un método para abordar la especificación de los requisitos del software basándose en modelos de comportamiento, escenarios de interacción del usuario con el sistema software, es decir, se basa en lo que el usuario espera del sistema. Utiliza técnicas estructuradas para la realización de los modelos oportunos. Por lo tanto, además de aportar un enfoque diferente en la construcción de sistemas software, este libro es otra fuente de referencia para las técnicas estructuradas así como un compendio de ejemplos prácticos.

Matheron, Jean-Patrick. "*Merise - Metodología de Desarrollo de Sistemas. Casos Prácticos*". Paraninfo, 1990.

Libro con ejercicios y soluciones planteadas bajo la metodología Merise. Supone una buena referencia para buscar ejemplos prácticos que, tras su previa adaptación al enfoque que se da en la asignatura, pueden servir de ejemplos en las clases teóricas, como ejercicios para las clases prácticas o de supuestos prácticos en los exámenes.

Miguel, Adoración de y Piattini, Mario G. "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al ModeloRelacional*". Ra-ma, 1993.

En este libro se explican las principales técnicas de modelado de datos, así como el proceso de diseño de bases de datos.

Miguel, Adoración de y Piattini, Mario. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

Relacionado con el presente tema se recomiendan los capítulos 3, **Concepto de modelo de datos** y 4, **El modelo entidad/interrelación**.

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Con relación a este tema puede resultar interesante estudiar la fase de análisis de sistemas dentro de la metodología Métrica 2.1, compuesta por los módulos de **análisis de requisitos del sistema** y de **especificación funcional del**

sistema. También se pueden ver las principales técnicas de modelado que se aplican en el análisis de requisitos (DFD, Modelado de datos, HVE...).

Pfleeger, Shari Lawrence. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

El capítulo 4 de este libro, **Capturing the requirements**, realiza una presentación del proceso de captura de requisitos. De las técnicas de especificación hace un rápido repaso de algunas (DFDs, SADT...), sin profundizar en ninguna. Lo más interesante es que plantea dos ejemplos, uno de gestión y otro de tiempo real.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

De este libro se destaca el capítulo 6: **Análisis de necesidades y estudio de viabilidad**, del cual merece una especial mención la exposición de las técnicas de recogida de información.

Sin embargo, el capítulo más interesante es el 7: **Análisis de sistemas**, donde se hace un amplio repaso por las diferentes técnicas de modelado en el análisis de requisitos, y que ha servido como referencia principal a la hora de desarrollar el presente tema.

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición, McGraw Hill, 1998.

Dentro de su tercera parte, **Métodos convencionales de la Ingeniería del Software**, se destacan tres capítulos: Capítulo 10: **Ingeniería de sistemas**; Capítulo 11: **Principios y conceptos de análisis**; Capítulo 12: **Modelado de análisis**.

Raghavan, S., Zelesnik, G. and Ford, G. "*Lecture Notes on Requirements Elicitation*". Educational Materials. CMU/SEI-94-EM-10. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March, 1994.

La obtención de requisitos es considerada por estos y otros autores como el primer paso en la Ingeniería de Requisitos (siendo las otras el análisis, la especificación y la validación). Este informe se centra en establecer una guía de referencia para la docencia relacionada con la obtención de requisitos.

Sawyer, Pete and Kotonya, Gerald. "*SWEBOK: Software Requirements Engineering Knowledge Area Description*". In [Abran et al., 1999], 1999.

Informe del SWEBOK sobre la Ingeniería de Requisitos.

SEI Requirements Engineering Project. "*Requirements Engineering and Analysis. Workshop Proceedings*". Technical Report CMU/SEI-91-TR-30 (ESD-TR-91-30). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). December, 1991.

Informe del *workshop* sobre Ingeniería de Requisitos y Análisis. Se tratan cuatro apartados: *Procesos y productos de la Ingeniería de Requisitos, Volatilidad de los requisitos, Obtención (elicitación) de requisitos* y *Técnicas y herramientas de la Ingeniería de Requisitos.*

Senn, James A. "*Análisis y Diseño de Sistemas de Información*". 2ª Edición. McGraw-Hill, 1992.

Libro dedicado a los sistemas de información, en el que se dedica una parte al análisis de requisitos.

Sibley, Edgar H. "*Entity-Life Modeling and Structured Analysis in Real-Time Software Design – A Comparison*". Communications of the ACM, 32(12):1458-1466. December, 1989.

Técnicas para el análisis de aplicaciones de tiempo real.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction & Tools*". Prentice-Hall, 1985.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 2: Essential Modeling Techniques*". Prentice-Hall, 1985.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 3:Implementation Modeling Techniques*". Yourdon Press, 1986.

Principios, técnicas y ejemplos para el modelado de sistemas software de tiempo real. Se define una extensión de la notación de Yourdon para sistemas de tiempo real.

Yourdon, Edward. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

Una referencia obligada a la hora de estudiar los DFDs. En el capítulo 9, **Diagrama de flujo de datos**, se presenta las características fundamentales de esta técnica. En el capítulo 10, **El diccionario de datos**, se estudia el uso, la importancia y la gramática de los diccionarios de datos. En el capítulo 11, **Especificaciones de proceso**, se incluyen las principales técnicas para la especificación de los procesos primitivos de los DFDs. El capítulo 13, **Diagramas de transición de estados**, se introducen estos diagramas, con la notación utilizada por Yourdon, así como su relación con otros modelos. Y el capítulo 14, **Balanceo de modelos**, expresa los pasos a dar para evitar las incongruencias entre los modelos realizados.

Tema 3: ***Análisis estructurado***

Descriptores

Análisis estructurado, enfoque clásico, métodos de los estímulos de Yourdon, modelo esencial, modelo ambiental, modelo de comportamiento, modelo de implantación, diagrama de contexto, diagrama de sistema, lista de acontecimientos.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T6** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño estructurado.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Señalar la diferencia entre la postura mantenida inicialmente por los partidarios de la descomposición funcional estricta, defendida sobretodo por el método de **Gane-Sarson** [Gane and Sarson, 1981], y el método de los estímulos que es sostenido por **Yourdon** [Yourdon, 1989].
- Familiarizar al alumno con la aplicación de un método sistemático en la aplicación de las técnicas estructuradas estudiadas en el tema anterior.

Contenidos

3.1 Introducción
3.2 Enfoque de modelado clásico
3.3 Enfoque de modelado de Yourdon

Tabla 5.10. Contenidos del tema 3 del temario teórico de Ingeniería del Software

Resumen

El presente tema se divide en tres apartados. En el primero de ellos se define el concepto de análisis estructurado, que surge como una consecuencia natural de los principios que inspiraron la programación estructurada.

El segundo apartado se presenta el enfoque clásico, totalmente descendente, con un enfoque de descomposición funcional estricto, con sus cuatro modelos (*el físico actual, el lógico actual, el lógico nuevo y el físico nuevo*) así como también sus problemas.

El tercer apartado se dedica al método de Yourdon, que parte de la lista de acontecimientos para construir un DFD preliminar al que aplicar un enfoque mixto ascendente/descendente.

Entre los apartados segundo y tercero se aprecia la evolución de los métodos estructurados de análisis.

Chris Gane y **Trish Sarson** han sido dos de los principales impulsores del denominado análisis estructurado estricto, basado en la descomposición funcional de

procesos bajo un planteamiento rigurosamente descendente (*top-down*), lo que se conoce con el nombre de **Análisis Estructurado Orientado a los Procesos**,. Los autores parten del diagrama de contexto, en el que se muestran las entidades externas relacionadas con el sistema, que se representa como todo, a través de los flujos de datos de entrada y salida. La explosión del diagrama de contexto en niveles sucesivos constituye la esencia del sistema. Un planteamiento semejante hace **Tom DeMarco** [DeMarco, 1979].

Las entidades externas y los flujos de datos que de ellas proceden, constituyen en el planteamiento de los autores citados, un punto de arranque suficiente para la descomposición del sistema en subsistemas. Por lo tanto, a partir del diagrama de contexto, mediante descomposición de los procesos aplicando las técnicas *del divide y vencerás* o de los refinamientos sucesivos, que se corresponden con los planteamientos descendentes, se va obteniendo el nivel de mayor detalle desde, y respetando la congruencia, los procesos dibujados en los diagramas de menor nivel de detalle.

Un aspecto interesante a considerar es el punto en el que deben aparecer los almacenes en los diagramas de flujos de datos. El criterio seguido es el de no incluirlos, hasta que un almacenamiento sea utilizado por más de un proceso.

El método Yourdon, también denominado método de los estímulos de **Edward Yourdon**, parte de la elaboración de una lista de eventos o estímulos que el sistema recibe de las entidades externas. Los estímulos provienen del exterior, o eventualmente de otros subsistemas relacionados con el que se está modelando, y se corresponden con los flujos de entrada. El sistema responde a cada uno de ellos mediante la realización de un proceso.

En sistemas complejos el método da lugar a la aparición de un gran número de procesos que son de difícil interpretación. Se realiza entonces lo que se denomina refinamiento ascendente (*upward leveling*), procedimiento consistente en agrupar los procesos afines en otro más general, ocultando en los procesos los almacenes correspondientes a los subprocesos que se agrupan. Este proceso es iterativo y se realiza tantas veces como sea necesario. El objetivo es evitar que existan en el diagrama un número excesivo de procesos. Esta es la visión ascendente (*bottom up*) del método Yourdon. Si alguno de los procesos que resultan de la lista de acontecimientos, no es primitivo, se puede refinar descendentemente, presentando el enfoque descendente de este método.

El método de Yourdon surge como crítica a los planteamientos de **Gane y Sarson**. **Yourdon** afirma pragmáticamente, que él no se opone al método, siempre que funcione, pero que en su opinión no es demasiado operativo, ya que cuando se analiza un sistema de información complejo y de gran tamaño, suele aparecer lo que se denomina la *parálisis en el sistema*. Afirma que el diagrama de contexto no representa una buena guía para la descomposición. Señala también que suele caer en el defecto de

descomponer el sistema en tantos subsistemas, como analistas estén disponibles, lo que implica una partición arbitraria.

El método de los estímulos resulta algo extraño desde un planteamiento puramente estructurado, ya que parece contradecir la propia esencia del paradigma, pero, visto objetivamente, puede parecer más avanzado y se acerca más a algunas de las técnicas empleadas posteriormente.

Bibliografía

- ***Citada en las transparencias del tema:***

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

- ***Lecturas complementarias:***

Miller, George A. "*The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*". The Psychological Review, Vol. 63: 81-97, 1956. Also available at http://www.well.com/user/smalin/miller.html [Última vez visitado, 3/8/1999].

Artículo clásico que se utiliza para poner un límite heurístico para la descomposición de un proceso en subprocesos, así como en otras técnicas de divide y vencerás.

Yourdon, Edward. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

Es una referencia obligada para este tema, tanto para la preparación de las clases por parte del docente, como para que los alumnos complementen la documentación facilitada. En concreto se recomienda la consulta de su tercera parte, **El proceso de análisis**, en la que aparecen los siguientes capítulos: **Capítulo 17 – El modelo esencial; Capítulo 18 – El modelo ambiental; Capítulo 19 – Construcción de un modelo preliminar de comportamiento; Capítulo 20 – Terminado del modelo de comportamiento; Capítulo 21 – El modelo de implantación del usuario.**

- ***Referencias utilizadas para preparar las clases:***

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto el tema 4, *Análisis de Requisitos del Software*, que en su parte final resume el método de Yourdon.

Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

La referencia más completa al método de Yourdon.

Tema 4: ***Diseño del software***

Descriptores

Diseño arquitectónico, diseño de datos, diseño de procedimientos, diseño de la interfaz, abstracción, refinamiento sucesivo, modularidad, arquitectura del software, estructura de programa, partición estructural, estructura de datos, procedimiento del software, ocultación de la información, diseño modular, módulo, complejidad ciclotómica, independencia modular, cohesión, acoplamiento.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T9** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Estudio y comprensión de los fundamentos del diseño de sistemas software. El diseño es una actividad fundamental en la construcción de cualquier artefacto, lo cual debe extenderse para el software, debiendo prestar a estos temas la suficiente atención en los currículos de Informática [Rasala, 1997], [Budgen, 1999].

De manera más concreta se pueden enunciar los siguientes objetivos:

- Conseguir que el alumno capte la importancia del diseño en la obtención de un software de calidad.
- Inculcar en el estudiante la necesidad de que los diseños deben respetar una serie de principios, como única forma de abordar un trabajo de forma profesional, alejado de la artesanía que tradicionalmente se venía utilizando en la construcción de los sistemas software.
- Buscar la independencia modular como criterio de calidad, desarrollando unos módulos altamente cohesionados, con bajo acoplamiento, y que hagan acopio del principio de ocultación de la información comunicándose a través de unas interfaces precisas y de pequeño tamaño.

Contenidos

4.1 Introducción
4.2 Proceso de diseño
4.3 Actividades de diseño
4.4 Fundamentos de diseño
4.5 Diseño modular

Tabla 5.11. Contenidos del tema 4 del programa teórico de Ingeniería del Software

Resumen

Este tema se ha divido en cinco apartados principales, intentando transmitir los principios y conceptos generales del diseño del software, necesarios para obtener software de calidad.

El primer apartado realiza una introducción que justifica la fase de diseño como necesaria para la calidad de los productos software, situándolo en el contexto del ciclo de vida. También se presenta la evolución del diseño desde la década de los setenta hasta el final de la década de los noventa.

En el segundo apartado se define el proceso de diseño y se enumeran las actividades que conlleva este proceso según diferentes autores. Ya en el tercer apartado se explican con un mayor grado de detalle las actividades relacionadas con el diseño arquitectónico, con el diseño de datos, con el diseño de los procedimientos y con el diseño de la interfaz.

El apartado cuatro se dedica a los fundamentos del diseño.

El primero de ellos, la *abstracción*, implica la descripción de una función de un programa en el nivel de detalle adecuado. Cada paso en el proceso de la Ingeniería del Software es un refinamiento del nivel de abstracción de la solución software (abstracciones funcionales y abstracciones de datos).

El *refinamiento sucesivo* es un concepto muy ligado al concepto de abstracción. Es una estrategia de diseño descendente por la que se va pasando de los niveles superiores de abstracción a los niveles inferiores, es decir, la manera en que se va añadiendo información de un nivel a otro [Wirth, 1971].

La *modularidad* implica que las funciones han de ser agrupadas, según su afinidad, en módulos, lo que facilita su mantenimiento.

La *arquitectura de software* hace referencia a la estructura jerárquica de los módulos, a sus interacciones y sus estructuras de datos. En un sentido más amplio se estaría refiriendo a la generalización de sus componentes.

La *estructura del programa* hace referencia a la organización jerárquica de los módulos basada en el flujo de control entre diferentes partes de un programa, no presentado detalles procedimentales.

La *partición estructural* establece que un programa debe partirse tanto horizontal como verticalmente. La partición horizontal define ramas separadas de la jerarquía modular para cada función principal del programa (*enfoque entrada/proceso/salida*). Por su parte la partición vertical expresa que el control, toma de decisiones, y el trabajo se distribuyan de forma descendente en la arquitectura del programa.

La *estructura de datos* representa la relación lógica existente entre los elementos individuales de datos, dictando la organización, los métodos de acceso, el grado de asociatividad y las alternativas de procesamiento para la información.

El *procedimiento del software* expresa los detalles de procesamiento de cada módulo individual, debiendo proporcionar una especificación precisa del procesamiento.

La *ocultación de la información* [Parnas, 1972] establece que cada módulo se caracteriza por decisiones de diseño que lo ocultan del resto de módulos. Los módulos deben especificarse y diseñarse cuidando que la información contenida dentro de ellos (*procedimientos y módulos*) no esté accesible para otros módulos. En general, la interfaz de un módulo debe revelar lo menos posible de su funcionamiento interno.

Por último, el quinto apartado se dedica al diseño modular, como base para buscar una independencia modular efectiva, esto es, la calidad del diseño, y del producto final en definitiva, está directamente relacionado con un conjunto de módulos lo más independientes posibles.

Existen dos medidas cualitativas de la independencia modular que son la *cohesión* y el *acoplamiento*.

La cohesión es la medida del grado de relación que guardan todas las sentencias incluidas en un módulo.

El acoplamiento es utilizado para medir el grado de independencia entre módulos, siendo el objetivo que se produzca en la menor medida posible.

Bibliografía

- ***Citada en las transparencias del tema:***

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Dahl et al., 1972] Dahl, Ole-Johan, Dijkstra, E. and Hoare, C. A. R. "*Structured Programming*". Academic Press, 1972.

[Date, 1995] Date, C. J. "*An Introduction to Database Systems*". 6th Edition, Addison-Wesley, 1995.

[Dennis, 1973] Dennis, J. "*Modularity*". In *Advanced Course on Software Engineering*, F. L. Bauer (editor), Springer-Verlag, pages 128-192, 1973.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Gamma et al., 1995] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd edition. Prentice Hall, 1997.

[Myers, 1978] Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978.

[Parnas, 1972] Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

[Peña, 1998] Peña Marí, Ricardo. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Ed. Prentice-Hall, 1998.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pressman, 1992] Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Shaw and Garlan, 1995] Shaw, M., and Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

[Sommerville, 1996] Sommerville, Ian. "*Software Engineering*". 5th Edition. Addison-Wesley, 1996.

[Stevens et al., 1974] Stevens, W.P., Myers G.J. and Constantine, L.L. "*Structured Design*". IBM Journal, 13(2):115-119, 1974.

[Taylor, 1959] Taylor, E. S. "*An Interim Report on Engineering Design*". Massachusetts Institute of Technology, 1959.

[Yourdon and Constantine, 1979] Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

[Warnier, 1974] Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

[Wasserman, 1983] Wasserman, A. "*Information Systems Design Methodology*". In *Software Design Techniques*. P. Freeman y A. Wasserman Editors., 4th Edition, IEEE Computer Society Press, 1983.

[Wirfs-Brock et al., 1990] Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wirth, 1971] Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

- ***Lecturas complementarias:***

Guttag, John. "*Abstract Data Types and the Development of Data Structures*". Communications of the ACM, 20(6):396-404. June, 1977.

Artículo que sirve de repaso al concepto de tipo abstracto de dato y su utilización para el diseño de las estructuras de datos. También es un ejemplo de la idea de que la abstracción y el refinamiento alcanzan tanto a los procesos como a los datos de un sistema software.

Hofmann, Hubert F., Pfeifer, Rolf and Vinkhuyzen, Erik. "*Situated Software Design*". Technical Report. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. 1993.

Informe que presenta algunos problemas en la concepción del proceso de diseño, planteando alguna aproximación para su solución.

Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

Artículo clásico donde David Parnas enuncia su afamado principio de ocultación de la información.

Perrochon, Louis and Mann, Walter. "*Inferred Designs*". IEEE Software, 16(5):46-51. September/October, 1999.

Artículo que destaca la importancia de la arquitectura software en el diseño de los sistemas software.

Rumbaugh, James. "*Bridging the Gap. Building Complex Systems by Leveling and Layering*". Rose Architect Magazine, 2(2). Winter, 1999.

Como combatir la complejidad en el diseño de grandes sistemas.

Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

Otro artículo clásico donde a través del problema de las ocho reinas, Niklaus Wirth va aplicando el principio de refinamiento sucesivo para obtener el diseño final.

- ***Referencias utilizadas para preparar las clases:***

Appleton, Bradford D. "*A Software Design Specification Template*". http://www.enteract.com/~bradapp/docs/sdd.html. 1997.

Estructura de un documento de especificación de diseño.

Bell, Doug, Morrey, Ian and Pugh, John. "*Software Engineering. A Programming Approach*". 2nd Edition. Prentice Hall, 1992.

Libro que más que de Ingeniería del Software trata de programación. Su parte II, **Design**, es la que más aplicación puede tener al enfoque que se le da a esta asignatura.

Budgen, David. "*Introduction to Software Design*". SEI Curriculum Module SEI-CM-2-2.1. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1989.

Material destinado a los docentes para la preparación de sus clases relacionadas con el diseño de software. Incluye temas a tratar y referencias bibliográficas comentadas.

Davis, William S. "*Herramientas CASE. Metodología Estructurada para el Desarrollo de los Sistemas*". Paraninfo, 1992.

Libro con un título engañoso, pues sólo dedica el capítulo 19, **Herramientas CASE**, y un apéndice, **Lista de productos CASE**, a la propia tecnología CASE (que por otra parte quedan un tanto desfasados al tratarse de la traducción al español de [Davis, 1983]), dedicándose el resto de capítulo a técnicas de modelado estructurado.

Fairley, Richard. "*Ingeniería del Software*". McGraw Hill, 1988.

Libro que fue referencia obligada para el establecimiento de los programas docentes de las asignaturas de Ingeniería del Software a finales de la década de los ochenta, principios de los noventa (es la traducción de [Fairley, 1985]). Sin embargo, la falta de nuevas ediciones lo han convertido en una obra casi obsoleta, donde el único servicio que ofrece es la descripción de alguna técnica que las ediciones de los libros actuales ya no contemplan.

En relación con la asignatura, su capítulo más aprovechable es el 5, **Diseño software**.

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto el tema 5, *Diseño del Software*, que recoge la explicación detallada de los fundamentos del diseño de sistemas software.

Hix, D. and Hartson, H. R. "*Developing User Interfaces: Insuring Usability through Product and Process*". John Wiley & Sons, 1993.

Este es un libro pensado para un curso completo de diseño de interfaces de usuario. En el se distinguen dos partes: el producto y el proceso. En los capítulos de la primera parte, el producto, se hace una descripción completa de los conceptos y los diferentes tipos de interfaz de usuario existentes. En la segunda parte, el proceso, se propone el ciclo de vida, los métodos y las notaciones de desarrollo de las interfaces de usuario.

Kaman Sciences Corporation. "*A State of the Art Report: Software Design Methods*". Kaman Sciences Corporation. March, 1994.

Como su propio nombre indica, se trata de un estado del arte sobre los métodos de diseño. No trata ninguno en profundidad.

Marqués Corral, José Manuel. "*Diseño de Interfaces de Usuario*". Apuntes de la asignatura Ingeniería del Software II de la Ingeniería Técnica de Informática de Sistemas. Universidad de Valladolid.

Tema dedicado al diseño de la interfaz de uusario.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999.

Este es un libro de recomendación prácticamente obligada en la tecnología de objetos, traducción de [Meyer, 1997], pero cuyos primeros capítulos pueden considerarse válidos para una asignatura de Ingeniería del Software, y con relación al proceso de diseño cabe destacar el capítulo 3, **Modularidad**.

Miguel, Adoración de y Piattini, Mario. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

Libro sobresaliente en el área de las bases de datos, sucesor de un libro ya clásico [Miguel y Piattini, 1993]. En relación con la asignatura se recomienda la lectura de su capítulo 8, **Diseño lógico de las bases de datos en el modelo relacional**.

Newman, M. and Lamming, G. "*Interactive System Design*". Addison-Wesley, 1995.

En este libro se aborda el diseño de la interfaz de usuario desde una perspectiva más amplia, el diseño de la interacción hombre-máquina, que según se propone debe formar parte principal de los procesos de Ingeniería del Software. Es muy interesante la idea que se expone acerca de la dependencia de la potencia y capacidad de utilización del sistema del diseño de la interfaz de usuario. Tiene una buena y amplia exposición acerca de las diferentes notaciones y los estilos de interacción en el diseño de interfaces de usuario.

Parnas, David L. "*Designing Software for Ease of Extension and Contraction*". IEEE Transactions on Software Engineering, SE-5(2):128-138. March, 1979.

Artículo en el que se aborda el diseño de software extensible.

Pressman, Roger S. "*Ingeniería del software: Un enfoque práctico*". 4ª Edición, McGraw Hill, 1998.

El proceso de diseño software está tratado en esta edición del libro de Pressman de una manera sobresaliente en los capítulos:

Capítulo 13 – Conceptos y Principios del Diseño. En este capítulo se abordan los principios básicos del diseño software.

Capítulo 14 – Métodos de Diseño. Capítulo dedicado a las diferentes facetas de diseño, destacándose el diseño arquitectónico.

Para ampliar conocimientos se recomiendan los capítulos:

Capítulo 15 – Diseño para Sistemas de Tiempo Real. Como complemento al diseño de aplicaciones de gestión.

Capítulo 21 – Diseño Orientado a Objetos. Como introducción al DOO.

Rivero Cornelio, E. "*Bases de Datos Relacionales*". Paraninfo, 1991.

Libro donde se trata la teoría de la normalización de bases de datos relacionales de una forma muy didáctica.

Shneiderman, B. "*Designing the User Interface. Strategies for Effective Human-Computer Interaction*". 2nd Edition. Addison-Wesley, 1992.

Se presenta un amplio estudio de los aspectos relacionados con el diseño, la implementación, la gestión, la formación y la mejora de la interfaz de usuario en los sistemas interactivos. Para la formación en diseño de interfaces de usuario se considera muy apropiado la exposición de los factores humanos a considerar en el diseño de sistemas interactivos (cap. 1), así como el tratamiento de los estilos de interacción soportados por menús (cap. 3), por lenguajes de comandos (cap. 4) y de manipulación directa (cap. 5).

Sommerville, Ian. "*Software Engineering*". 5th Edition, Addison-Wesley, 1996.

Otro de los libros clásicos de Ingeniería del Software, que dedica su Parte III al diseño del software (Capítulos 12, 13, 14, 15, 16 y 17) haciendo un recorrido bastante amplio por los diferentes métodos de diseño.

Tremblay, Guy. "*Knowledge Area Description for Design (version 0.5)*". In [Abran et al., 1999], 1999.

Informe del SWEBOK sobre el diseño del software.

Tema 5: ***Diseño estructurado***

Descriptores

Diagrama de estructuras, tabla de interfaz, estrategias de diseño, análisis de transformación, análisis de transacción, centro de transformación, centro de transacción.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T6** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño estructurado.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Estudiar la transformación de los modelos realizados en análisis a sus correspondientes en diseño. En el caso particular del tema de los DFDs a los diagramas de estructuras y del modelo entidad/relación a un modelo relacional normalizado.
- Distinguir entre las estrategias de diseño de transformación y transacción a la hora de transformar los DFDs en diagramas de estructuras.

Contenidos

5.1 Introducción
5.2 Diagrama de estructuras
5.3 Estrategias de diseño

Tabla 5.12. Contenidos del tema 5 del programa de teoría de Ingeniería del Software

Resumen

El objetivo principal de este tema es explicar la forma correcta de pasar del análisis al diseño, haciendo hincapié en el uso de las denominados diagramas de estructuras que provienen de los diagramas de flujo de datos, es decir en el diseño arquitectónico. Esto es así porque se entiende que el alumno ya está familiarizado de forma práctica con el resto de las actividades de diseño. Así, en la asignatura de **Diseño de Bases de Datos** ha estudiado el paso del modelo conceptual al modelo lógico de datos y la teoría de la normalización, en la asignatura de **Interfaces Gráficas** ha profundizado en el diseño de interfaces gráficas de usuario, y en las asignaturas de programación ha practicado el diseño de algoritmos previamente a su codificación.

En el primero de los tres apartados en que se organiza el tema se hace una introducción al diseño estructurado, enlazando con lo estudiado en el tema anterior, y justificando (tal como se ha hecho en el párrafo previo) la orientación del mismo hacia el diseño arquitectónico.

En el segundo apartado se estudia la técnica fundamental del diseño arquitectónico dentro del paradigma estructurado, los *diagramas de estructura*, también conocidos por el término en inglés *structure chart*, o por *diagrama de estructura de cuadros de Constantine* [MAP, 1995].

Estos diagramas se basan en la estructura jerárquica de los módulos; cada uno de los cuales tiene una función propia, y además se comunica sólo con su módulo *superior,* del que recibe órdenes, y con sus módulos *subordinados*, a los que da órdenes.

En la tercera, y última, parte de este tema se estudian las estrategias de diseño, que son aquéllas que se emplean para la transformación del modelo funcional basado en DFDs en los diagramas de estructuras que constituyen el diseño arquitectónico.

Lo primero sobre lo que se llama la atención es lo *artificial* de este proceso, que obliga a cambiar de modelo subyacente en la técnica de modelado: *de los procesos del DFD* se pasa *a la jerarquía de módulos*.

Se distinguen dos tipos de flujos de datos: *el flujo de transformación* y *el flujo de transacción*.

El flujo de transformación reproduce fielmente el patrón entrada/proceso/salida, de manera que las decisiones las toman los módulos de ámbito superior y las operaciones se llevan a cabo por los módulos de orden inferior.

El flujo de transacción establece dos módulos principales, uno que analiza la transacción a ser tratada y otro que encamina hacia el módulo que la trata. En estos sistemas se tienen diferentes caminos de acción, independientes los unos de los otros, de forma que la transacción servirá de elemento discriminador para determinar el camino a elegir.

La existencia de dos tipos de flujos de datos origina dos estrategias de diseño, la primera de ellas, el análisis de transformación, se aplica en aquellos sistemas donde la característica fundamental sea de flujo de trasformación, mientras que la segunda, el análisis de la transacción, se aplica en los sistemas cuya característica principal sea de transacción.

Bibliografía

- ***Citada en las transparencias del tema:***

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Yourdon and Constantine, 1979] Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.

- ***Lecturas complementarias:***

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Es interesante ver cómo se aplican las diferentes estrategias de diseño dentro del contexto de una metodología concreta como puede ser Métrica 2.1.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

El capítulo 8 de este libro, **Diseño Estructurado de Sistemas**, está dedicado casi por completo al diseño estructurado de sistemas, con algunas referencias al diseño de datos y a las metodologías de diseño detallado de programas. Con el complemento de los ejercicios planteados al final del mismo, se convierte en una buena referencia para el estudio de este tema.

- ***Referencias utilizadas para preparar las clases:***

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto el tema 5, *Diseño del Software*, dado que el final del mismo está dedicado a las estrategias de diseño: análisis de transformación y de transacción.

Pressman, Roger S. "Ingeniería del software: Un enfoque práctico", 4ª Edición, McGraw Hill, 1998.

Su capítulo 14, **Métodos de diseño**, contiene un apartado completamente dedicado a las estrategias del diseño arquitectónico estructurado.

Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.

Referencia clásica sobre diseño estructurado.

Unidad Docente III: Introducción al paradigma objetual

Objetivo genérico

El objetivo que se plantea esta unidad docente es la introducción al alumno en el enfoque de análisis y diseño orientado a objetos para el desarrollo de software, cuyo fin es la construcción de modelos de objetos computacionalmente implantables, obtenidos por observación del mundo real, respetando y reproduciendo sus agentes y vínculos.

Aunque el mayor esfuerzo dentro de este paradigma se ha hecho en los lenguajes de programación, es, sin embargo, en las fases de análisis y diseño donde su empleo puede dar lugar a unos resultados más brillantes.

Es importante insistir en que una buena técnica de diseño retrasa todo lo posible los detalles de implementación, ya que de esta forma se gana flexibilidad. Lo que realmente importa es definir el problema, captar sus requisitos y ser capaces de plantear una solución a las necesidades de información, que se acople de la manera más natural posible a la organización y a las responsabilidades de los usuarios.

También debe resaltarse que es esencial evitar los errores en las primeras fases del desarrollo, porque su influencia sobre la calidad del sistema es fundamental.

Debe presentarse, como uno de los aspectos más importantes del nuevo paradigma, que supone una forma de pensar distinta acerca del problema que se trata de resolver, incorporando de una forma más natural las característica relevantes de los objetos del mundo real. Se trata en suma de modelar los objetos del negocio, o del domino del problema, es decir los conceptos familiares a los usuarios. Estos conceptos no sólo implican estructura de datos, sino que además llevan asociado el comportamiento, que representa la responsabilidad de la entidad objetualizada ante los vínculos con otros objetos. Por tanto es el usuario el que, como conocedor de la realidad del dominio del problema, aclare el comportamiento que se espera que tengan los objetos del modelo.

Ahondando en el punto anterior, y desde la perspectiva del análisis orientado a objetos, lo más importante es plantearse el problema bajo el enfoque del usuario, y no bajo el planteamiento meramente informático o tecnológico. Esto significa que el esfuerzo en la tarea del desarrollo del software debe hacerse más en la identificación de los objetos pertenecientes al dominio del problema, que en la definición de los objetos relacionados con la aplicación desde el punto de vista de su entorno de implementación.

El paso al diseño y a la implementación es más natural que en el paradigma estructurado, simplemente por el mero hecho de que el modelo subyacente a todas las fases del ciclo vital del software es el mismo, el modelo objeto, llegándose a lo que se denomina un proceso de construcción de software sin costuras [Nerson, 1992]. Así, el paso de los modelos de análisis a los de diseño no conllevará una transformación, como en el paradigma estructurado, sino que se hará por elaboración de los primeros, esto es,

los objetos del dominio del problema se refinan, perfilando más sus características, y se añaden los objetos de interfaz, de utilidad y de aplicación [Monarchi and Puhr, 1992].

La inclusión de esta unidad docente en el programa de la asignatura de Ingeniería del Software está plenamente justificada desde diversos puntos de vista:

- Por estar los métodos de desarrollo de software orientados a objetos contemplados en los diferentes cuerpos de conocimiento sobre Ingeniería del Software que se han tenido en cuenta.
- Por ser muchas las referencias que abogan por un enfoque orientado a objetos para la asignatura de Ingeniería del Software [Donadi, 1992], [Jacobson et al., 1993], [Bruegge and Dutoit, 2000].
- Por la evolución que están sufriendo los métodos y las herramientas de desarrollo, cada día más asentadas sobre la tecnología de objetos, soportando los ciclos de vida iterativos e incrementales, de desarrollo rápido de aplicaciones, paradigmas visuales... todo ello bastante alejado del paradigma estructurado.

Esta unidad docente ocupa el 29% del temario de teoría de la asignatura, estando compuesta por tres temas (**Introducción a la orientación a objeto**, **UML** y **Visión general de la metodología OMT**), que se detallan a continuación.

Tema 6: ***Introducción a la Orientación a Objetos***

Descriptores

Tecnología de objetos, reutilización del software, modelo objeto, abstracción, encapsulamiento, modularidad, jerarquía, paso de mensajes, tipo, polimorfismo, clase, objeto, atributo, método, estado, comportamiento, identidad, análisis orientado a objetos, diseño orientado a objetos.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T3** y **T7** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en los conceptos fundamentales de la tecnología de objetos, explicándole las diferencias que presenta esta manera de desarrollar software respecto a los planteamientos estructurados.
- Insistir en que los objetos representan elementos del mundo real dentro del dominio del problema que se debe resolver, siendo por tanto, familiares al usuario. Los objetos llevan incorporado el comportamiento habitual que el experto en el dominio espera de ellos.
- Destacar dentro de los conceptos fundamentales, el especial relieve que tienen para la obtención de un software de calidad los conceptos de clasificación (*jerarquía*) y encapsulación.
- Señalar el aumento de productividad que se alcanza gracias al polimorfismo y la herencia, pero también el significado avanzado de estos conceptos en las fases de análisis y diseño.
- Recalcar la trascendencia de la Orientación a Objetos en el análisis de aplicaciones, ya que su faceta más conocida es en la fase de codificación.
- Comentar y discutir los conceptos básicos del *modelo objeto* basándose en las aportaciones de varios autores, y el modo en el que pueden soslayarse las posibles limitaciones tanto conceptuales, como de notación.
- Introducir el concepto de reutilización del software, su importancia para el aumento de la productividad en el desarrollo del software y su relación con la Orientación a Objetos. En este sentido hacer hincapié en que la reutilización del software está presente en todo el ciclo de vida, no sólo en la implementación, debiendo empezar a pensar en ella en las primeras fases del ciclo de vida.

Contenidos

6.1 Introducción y evolución de la Orientación a Objetos
6.2 Modelo objeto
6.3 Análisis y diseño orientados a objetos

Tabla 5.13. Contenidos del sexto tema del programa de teoría de Ingeniería del Software

Resumen

Este tema, como se aprecia en la Tabla 5.13, se divide en tres apartados principales. El primero de ellos sirve como introducción de la tecnología de objetos. Aunque a lo largo de la asignatura ya se ha hecho mención al paradigma objetual (*y los alumnos han manejado de forma intuitiva los objetos en la asignatura de Interfaces Gráficas de segundo curso*), este apartado supone la "presentación oficial" de la Orientación a Objeto en su currículo.

En este primer apartado se justifica el auge de la tecnología de objetos para que, con una manera diferente de enfocar el desarrollo del software, se pueda hacer frente a la continua demanda social de software de mayor calidad, complejidad y sencillez de manejo.

Se discuten los beneficios potenciales, las ventajas y los inconvenientes (mitos mayormente) y peligros de la Orientación a Objetos. Dentro de los beneficios que se pueden lograr con el uso de la tecnología de objetos se hace mucho hincapié en la reutilización del software, primero para resaltar su importancia en la consecución de sistemas de software de calidad realizados con alta productividad, segundo para aclarar que la reutilización del software es ortogonal a la Orientación a Objetos (y a cualquier otro método de desarrollo) aunque este paradigma facilita enormemente tanto el *desarrollo para reutilización* como el *desarrollo con reutilización*, y tercero que la reutilización no es un fin sí misma, sino una forma de enfrentarse de una forma competitiva al desarrollo de software de calidad [García, 2000].

El segundo apartado se dedica a la presentación del modelo objeto, como denominador común de todos los conceptos y principios que debe cumplir un paradigma de desarrollo para ser considerado orientado a objetos. Aunque no existe un acuerdo completo entre los especialistas, se va a exponer lo que, a juicio de **Grady Booch** [Booch, 1994], constituyen los elementos fundamentales de un modelo objeto. En la explicación de los mismos tendrán cabida diferentes perspectivas, aunque predominan las cercanas al modelo objeto subyacente en UML [OMG, 1999], al ser este un lenguaje de modelado considerado como estándar por OMG, y que será objeto de estudio en el tema 7 de esta asignatura.

El tercer apartado se dedica a introducir las fases de análisis y diseño en los desarrollos orientados a objetos.

Se enfatiza como la fase de obtención de requisitos es independiente de si se va a desarrollar orientado a objetos o estructurado, igual que el énfasis de la fase de análisis sigue siendo el *qué hacer* y la misión del diseño modelar el *cómo hacerlo*.

En la Orientación a Objetos el software se organiza como una colección de objetos discretos, los cuales contienen datos y comportamiento; manteniéndose este modelo a lo largo de todo el ciclo de vida, lo que permite una transición mucho más suave entre las diferentes fases de éste, a la vez que se facilita la utilización de modelos de ciclo de vida iterativos e incrementales.

De forma explícita se comentan diversas actividades a realizar en las fases de análisis y diseño para favorecer el desarrollo para reutilización, lo que se presenta como una máxima para todo ingeniero del software.

Bibliografía

- ***Citada en las transparencias del tema:***

[Berard, 1996] Berard, Edward V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc., 1996.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Budd, 1991] Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Champeaux et al., 1993] Champeaux, Dennis, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison Wesley, 1993.

[Freeman, 1987] Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman). Pages 2-8. IEEE Computer Society Press, 1987.

[García et al., 1997] García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel. "*Mecano: Una Propuesta de Componente Software Reutilizable*". En las actas de las II Jornadas de Ingeniería del Software (Donostia-San Sebastián, España, 3-5 de septiembre de 1997): 232-244. 1997.

[García y Pardo, 1998] García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Introducción al Análisis y Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(2): 64-70. Febrero, 1998.

[Girow, 1996a] Girow, Andrew. "*Objects and Binary Relations*". Object Currents, 1(6), SIGS Publications, June 1996.

[Girow, 1996b] Girow, Andrew. "*Binary Relations Approach to Building Object Database Model*". Object Currents, 1(11), SIGS Publications, November 1996.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

[Joyanes, 1998] Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. Osborne McGraw-Hill. 1998.

[Krueger, 1992] Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

[Liskov, 1988] Liskov, B. "*Data Abstraction and Hierarchy*". SIGPLAN Notices, Vol. 23(5), 1988.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd edition. Prentice Hall, 1997.

[Monarchi and Puhr, 1992] Monarchi, David E. and Puhr, Gretchen I. "*A Research Typology for Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):35-47. September, 1992.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Parnas, 1972] Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

[Pepper, 1997] Pepper, Jon. "*Object Magazine Survey: What's Corporate America Spending on Objects?*". Object Magazine, February, 1997.

[Piattini, 1996] Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Rumbaugh et al. 1991] Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Smith and Tockey, 1988] Smith, M. and Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

[Sutherland, 1997] Sutherland, Jeff. "*Object World Tutorial - Object Design Tutorial*". 1997.

- ***Lecturas complementarias:***

Booch, Grady. "*Objectifying Information Technology*". Rational Software Corporation. http://www.rational.com [Última vez visitado, 1-12-1998]. 1998.

Artículo en el que Grady Booch expone su visión de las barreras que tiene la Orientación al Objeto para ser adoptada por las organizaciones.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 29 de noviembre de 1999.

Definición de las tareas a realizar, los productos a obtener y las técnicas a emplear durante la actividad de análisis de requisitos de la fase de ingeniería de requisitos del ciclo de vida de la Ingeniería del Software, tomando como paradigma de desarrollo la Orientación a Objeto.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Introducción al Análisis y Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(2):64-70. Febrero, 1998.

Artículo que presenta un somero repaso por los conceptos de ADOO.

Kaindl, Hermann. "*Difficulties in the Transition from OO Analysis to Design*". IEEE Software, 16(5):94-102. September/October, 1999.

Interesantísimo artículo que defiende que la transición entre los modelos de análisis y diseño no está exenta de problemas, precisamente por la dificultad de determinar en qué fase se está en ciertos momentos y porque el significado de los objetos de análisis (*objetos semánticos*) y objetos de diseño no es la misma.

Lewis, Ted. "*The Dark Side of Objects*". IEEE Computer, 27(12):6-7. December, 1994.

Artículo de opinión en la que el autor denota el cambio en los productos de desarrollo de software hacia la Orientación a Objetos, de manera que si un producto no hace referencia a este paradigma en su nombre pierde cuota de mercado.

Rine, David. "*Object-Oriented Technology and Software Reuse*". IEEE Computer, 26(7):6. July, 1993.

Breve artículo que trata la relación existente entre la Orientación a Objetos y la reutilización del software.

- ***Referencias utilizadas para preparar las clases:***

Booch, Grady. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996.

Libro fundamental en la Orientación a Objetos, traducción al castellano de [Booch, 1994]. Está orientado al diseño, muy sustentado en C++ para presentar sus ejemplos. Los capítulos de su primera parte, **Conceptos**, (**Capítulo 1: Complejidad; Capítulo 2: El modelo de objetos; Capítulo 3: Clases y objetos; Capítulo 4: Clasificación**) han sido las directrices sobre las que se ha fundamentado el apartado dos del presente tema.

Champeaux, Dennis de, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison-Wesley, 1993. HTML Edition available at http://gee.cd.oswego.edu/dl/oosdw3. [Última vez visitado, 1-2-2000].

Libro que se encuentra dividido en dos partes. La primera de ellas, **Analysis**, consta de 14 capítulos en los que se introduce el modelo objeto y el proceso de análisis. La segunda parte, **Design**, compuesta de 12 capítulos donde se aborda el paso del análisis a diseño y del diseño a la implementación.

Graham, Ian. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Traducción de [Graham, 1994]. Es un libro típico para introducirse en la Orientación al Objeto. Cada capítulo viene a ser un estado del arte de la Orientación a Objetos en la fecha de edición.

Henderson-Sellers, Brian. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Un libro muy interesante, escrito por uno de los mayores expertos en la tecnología de objetos. En relación con el presente tema se recomiendan los capítulos 2: **An introduction to the object-oriented philosophy and terminology**; 3: **Object-Oriented software engineering**; y 5: **Object modeling**.

Martin, James and Odell, James J. "*Métodos Orientados a Objetos: Conceptos Fundamentales*". Prentice Hall, 1997.

Es la traducción de [Martin and Odell, 1995]. Es un libro de referencia para el profesor porque su enfoque difiere del de otros autores, existiendo algunas diferencias conceptuales con UML. La notación seguida en los diagramas de clases recuerda a un modelo entidad-relación extendido. Uno de sus apartados más interesantes es en el que trata las relaciones de meronima (*Capítulo 18. Composición*) y las de tipo-subtipo (*Capítulo 9. Subtipos y supertipos: Primera parte; Capítulo 10. Subtipos y supertipos: Segunda parte*).

Se tiene constancia de la existencia de una segunda edición de este libro [Martín and Odell, 1998], pero no se ha tenido oportunidad de consultarlo.

Martin, James and Odell, James J. "*Métodos Orientados a Objetos: Consideraciones Prácticas*". Prentice Hall Hispanoamericana, 1997.

Traducción de [Martin and Odell, 1996]. Libro muy interesante en el sentido de que expone como la tecnología de objetos se pone en práctica en diferentes campos.

Nerson, Jean-Marc "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September, 1992.

Artículo introductorio al ADOO. Hace hincapié especialmente a la escasa separación entre las fases de desarrollo del software en la OO, lo que él denomina un proceso *sin costuras*.

Oestereich, Bernd. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

La primera parte de este libro, **Introduction**, presenta una introducción a la Orientación a Objeto muy amena, especialmente los dos primeros capítulos (**Introduction** y **Object-Orientation for beginners**).

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Transparencias de un curso de Orientación a Objetos.

Piattini, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Su capítulo 10, **Análisis y diseño orientado al objeto**, es una breve presentación de la tecnología de objetos, muy fácil y rápida de leer, muy recomendable a los alumnos como lectura previa a la exposición del tema.

Robinson, Peter J. "*Hierarchical Object-Oriented Design*". Object-Oriented Series. Prentice-Hall, 1992.

Descripción del método de diseño HOOD (*Hierarchical Object-Oriented Design*), totalmente orientado al lenguaje Ada.

Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Otra referencia imprescindible en un tema de introducción a la Orientación a Objetos, especialmente relacionados con el tema se tienen el capítulo 1, **Introducción**, el capítulo 3, **Modelado de objetos** y el capítulo 4, **Modelado avanzado de objetos**.

Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice Hall, 1990.

Es otro libro clásico, donde los autores liderados por Rebecca Wirfs-Brock, desarrollan su metodología RDD basada en las responsabilidades y la colaboraciones de los objetos. Hay que destacar de este libro la utilización de las tarjetas CRC (*Class, Responsability and Collaboration*) en todo el ciclo de vida del desarrollo.

Yourdon, Edward and Argila, Carl. "*Case Studies in Object Oriented Analysis & Design*". Yourdon Press Computing Series. Prentice Hall, 1996.

Libro que se dedica más a presentar ejemplos de la aplicació de las técnicas de ADOO que de establecer los principios fundamentales de esta disciplina.

Yourdon, Edward, Whitehead, Katharine, Toman, Jim, Oppel, Karin and Nevermann, Peter. "*Mainstream Objects. An Analysis and Design Approach for Business*". Yourdon Press, 1995.

Libro en el que se describe un proceso de desarrollo de software orientado al objeto, creado por los autores para la empresa Software AG. De las cuatro partes con que cuenta este libro, la primera, **Introduction to Object Orientation**, es la que más se ajusta a los contenidos del presente tema, aunque como ampliación puede servir la parte III, **The Object-Oriented Development Process**, y la parte IV, **Thinking Object-Oriented: Analysis and Design Guidelines**.

Tema 7: *UML*

Descriptores

UML, lenguaje de modelado, modelo, vista, diagrama, elemento de modelado, diagrama estático de estructura, diagrama de clase, clase, atributo, método, asociación, adorno de asociación, cardinalidad, cualificador, agregación, composición, generalización/especialización, paquete, diagrama de interacción, diagrama de secuencia, diagrama de colaboración, mensaje, caso de uso, actor, relación entre casos de uso.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Que el alumno entienda qué es y qué no es UML, conozca someramente sus orígenes y la historia de la evolución metodológica de la Orientación a Objetos en las áreas de análisis y diseño.
- Que el alumno tome un contacto inicial con el vocabulario, las reglas, formas de empleo de UML y, en general, enfoque el manejo de la simbología y el alcance de los conceptos adecuadamente.
- Que el alumno se inicie en el modo de aplicar los modelos de UML para resolver los problemas implícitos en el desarrollo del software.
- Que el alumno domine el modelado estructural a un nivel básico.
- Introducir el concepto de caso de uso y de su importancia en las corrientes metodológicas actuales, bien como herramienta de extracción y documentación de los requisitos del sistema software, bien como herramienta para modelar el comportamiento esencial de un sistema o subsistema en conjunción con otros diagramas (*colaboración, secuencia, transición de estados...*).

Contenidos

7.1 Introducción
7.2 Una visión general de UML
7.3 Diagramas de estructura
7.4 Diagramas de interacción
7.5 Casos de uso

Tabla 5.14. Contenidos del séptimo tema del programa de teoría de Ingeniería del Software

Resumen

El tema se ha dividido en cinco apartados principales, que tienen como cometido introducir al alumno en el modelado orientado a objetos a través de UML. No se pretende que el alumno obtenga un dominio completo de UML, porque para ello habría que dedicar prácticamente la totalidad de la asignatura a este tema, sino que perciba los conceptos básicos, basados en los diagramas de clase, de casos de uso y de secuencia. Posteriormente, en la asignatura **Análisis de Sistemas**, del primer curso del segundo ciclo, se completarán los conceptos aquí introducidos.

El primer apartado, como bien indica su nombre, es una introducción a los orígenes y cometido de UML.

Primeramente se vuelve a incidir en algo que ya se había explicado en el tema 2, *¿Por qué hay que realizar modelos?*, pero ahora desde el prisma de los modelos realizados en el paradigma orientado a objeto, destacando sus características, especialmente su proximidad al modelado de la realidad tal cual es.

Se justifica el porqué de la necesidad de un estándar dentro de los lenguajes de modelado, UML en este caso, desde el punto de vista de lo que supone para las organizaciones donde se desarrolla software.

Así, una organización eficiente dentro del mundo de la Informática es aquélla que, respetando las directrices de la calidad del software, cubre las peticiones y necesidades de los usuarios. Una organización que pueda desarrollar el software en el tiempo y predeterminar resultados satisfactorios, con un empleo y una utilización eficiente de recursos, sostiene un *negocio* estable.

Para desarrollar software de calidad, es necesario disponer de una arquitectura de bases sólidas, que resistan el cambio. Para desarrollar software de un modo rápido, eficiente y efectivo, con un mínimo de pérdidas de código y trabajo, hay que disponer del personal, las herramientas y la orientación correcta.

Por otro lado, para trabajar en el desarrollo de software de un modo consistente, y considerando los costes del ciclo de vida del sistema, es básico poder adaptarse al cambio de las necesidades de las empresas y negocios, así como a la evolución de la tecnología.

El modelado riguroso y acertado adquiere una importancia trascendental ante este panorama. Se tienen que construir modelos que trasladen pronto la estructura y el comportamiento que se plantea para un sistema. Se deben orientar los modelos hacia la visualización y el control de la arquitectura del sistema. Se tienen que diseñar modelos que representen adecuadamente el sistema que se necesita lograr, y que frecuentemente, ofrezcan posibilidades de simplificación y reutilización. Hay que conseguir modelos para gestionar el riesgo.

UML se justifica como una propuesta acertada, porque su filosofía es modelar para entender lo mejor posible el sistema que se va a desarrollar. Con UML se consiguen las siguientes ventajas:

- UML ayuda a expresar un sistema tal como es, o como se quiere que sea.
- UML permite especificar la estructura y los comportamientos del sistema.
- UML aporta una pauta de desarrollo eficaz, que se muestra como una buena guía de los trabajos.
- UML documenta las decisiones que se toman al modelar y desarrollar.

El modelado no es exclusivo para los grandes sistemas [Moitra, 1999]. Sin embargo, es una verdad absoluta que el sistema más grande y complejo conlleva el más importante esfuerzo de modelado, y esto es así porque se *construyen modelos de los sistemas complejos ante la incapacidad humana de entender completamente como son éstos*.

Existen límites para la capacidad humana de entender la complejidad. Hay que plantear los sistemas estudiándolos y dirigiendo el esfuerzo sólo sobre un aspecto en cada caso. Esto es esencialmente el principio general de *divide y vencerás*, que **Dijkstra** entendió útil para el planteamiento de algoritmos ya hace años cuando dijo: "*abordar un problema duro dividiéndolo en una serie de pequeños problemas que se puedan resolver*".

El modelado de sistemas en la Ingeniería Informática necesita un lenguaje común, de la misma manera que otras ingenierías disponen de lenguajes *normalizados*. Así, existen lenguajes para la industria de la construcción, la industria eléctrica... Los matemáticos disponen de un lenguaje universal, y además formal, de modelado. Pero no es una cuestión del grado de formalismo del modelo (*muchos esquemas informales resultan adecuados en su aspecto expresivo para los fines de diseño y descripción que con ellos se persiguen*). Es más importante la necesidad de resolver una carencia que tenían las actividades de modelado en el ámbito informático: *las técnicas empleadas carecían de un lenguaje estándar*.

Existen cuatro principios de modelado, que deben ser respetados:

- La elección del modelo que va a ser creado tiene una influencia profunda sobre cómo se aborda el problema y se plantea la solución.
- Cada modelo puede ser expresado con diferentes niveles de precisión.
- Los mejores modelos están estrechamente ligados con la realidad.
- Un modelo único no es suficiente. Cada sistema no trivial se modela mejor a través de un pequeño conjunto de modelos relacionados.

UML es un **lenguaje**, porque tiene un vocabulario y unas reglas para combinar sus términos con el propósito de lograr la comunicación. Es un **lenguaje de modelado**,

porque su vocabulario y sus reglas se dirigen a la representación conceptual y física de un sistema. UML es un **estándar** porque constituye una forma común de expresar las especificaciones para el software, habiéndose admitido como tal por la comunidad de la Orientación a Objetos (*estándar de facto*) y porque ha sido reconocido de esta manera por un organismo internacional cualificado como es OMG (*estándar oficial*).

En resumen se puede decir que UML es:

- *Un lenguaje para visualizar:* en el sentido de ganar expresividad en lo que se diseña y se hace.
- *Un lenguaje para especificar:* en este contexto especificar significa construir modelos que sean precisos, no ambiguos y completos.
- *Un lenguaje para construir:* UML no es un lenguaje de programación visual, pero sus modelos pueden ser inmediatamente conectados con una amplia variedad de lenguajes de programación. Esto significa que es posible traducir desde un lenguaje como UML a un código Java, C++, Visual BASIC..., así como también a las tablas de una base de datos relacional u orientada a objetos, y hacerlo con el soporte automático de una herramienta CASE, como puede ser el caso de **Rational Rose 98** o **Rational 2000**.
- *Un lenguaje para documentar:* Los modelos realizados en UML son la base para la documentación del sistema software.

También puede decirse que UML no es:

- *Un lenguaje de programación visual.*
- *La especificación de una herramienta o de un repositorio.*
- *Una metodología, método o proceso software.*

En lo tocante a la evolución histórica de UML puede decirse que los lenguajes de modelado orientados a objetos hacen su aparición entre la mitad de la década de los setenta y finales de la década de los ochenta, con una orientación claramente metodológica, conjugada con un nuevo género de lenguajes de programación orientados a objetos e impulsada por el incremento constante de la complejidad de las aplicaciones.

Los metodólogos, superando su exclusividad para la programación, comienzan la aproximación del paradigma objetual a las tareas del análisis y diseño. El número de métodos crece desde poco más de diez en el año 1989 a más de 50 en 1994 [García y Pardo, 1998].

Muchos usuarios interesados en la Orientación a Objeto, no encuentran un método que cubra completamente sus necesidades, lo que motiva la aparición de métodos de mayor entidad metodológica, normalmente resultado de la evolución o de la fusión de los ya existentes.

Mediada la década de los noventa, los responsables directos de tres de los métodos de desarrollo orientado a objetos de mayor difusión, **Grady Booch, James Rumbaugh** e **Ivar Jacobson**, unen sus esfuerzos para unificar sus métodos, comenzando por la definición de un lenguaje de modelado[23], atendiendo a tres razones:

- Los métodos ya se estaban mezclando. Era más sensato continuar con su conjunta que por separado.
- Para unificar los métodos debían de alcanzar cierta estabilidad en el mercado, aportando para los proyectos un lenguaje de modelado maduro, que estimulara a los constructores de herramientas a incluir las características más útiles de éstos.
- Con la colaboración necesaria para la unificación, mejorarían todos, aprendiendo y compartiendo experiencias.

Para orientar la unificación se establecieron tres objetivos:

- Modelar sistemas desde el concepto de *artefacto ejecutable*, utilizando las técnicas orientadas a objetos.
- Dirigir los esfuerzos hacia el manejo de la complejidad, misión crítica de los sistemas.
- Crear un lenguaje de modelado unificado, utilizable tanto por las personas, como por los computadores.

Oficialmente los trabajos para crear UML comienzan en 1994. Al comienzo el proyecto se centró en la unificación de los métodos de **Booch** [Booch, 1994] y OMT [Rumbaugh et al., 1991], saliendo a la luz la versión 0.8 del llamado Método Unificando en octubre de 1995 [Booch and Rumbaugh, 1995]. Con la incorporación de **Ivar Jacobson**, se pospone la definición del método, poniendo el énfasis en el lenguaje de modelado, surgiendo la versión 0.9 de UML en junio de 1996 [Booch et al., 1996], la primera versión de UML. En noviembre de 1997, la versión 1.1 de UML [Rational et al., 1997] se adopta como estándar por OMG, siendo la versión 1.3 la que actualmente se encuentra en vigor [OMG, 1999].

En el apartado segundo del tema se hace un recorrido rápido por UML, diferenciándose las siguientes partes:

- ***Vistas:*** Que muestran diferentes aspectos del sistema que se está modelando. No es algo gráfico, sino una abstracción compuesta de diversos diagramas. Las vistas enlazan el lenguaje de modelado con el método/proceso elegido para el desarrollo
- ***Diagramas:*** Son grafos que describen los contenidos de una vista

[23] La evolución histórica de los métodos de desarrollo en la Orientación a Objetos se presentó con más detalle en el apartado 4.3.2.3 de este mismo Proyecto Docente.

- ***Elementos de Modelado:*** Son los conceptos utilizados en los diagramas, que representan los conceptos del paradigma objetual (clases, objetos, relaciones...). Un elemento de modelado puede estar en diferentes diagramas, pero siempre con el mismo significado y símbolo asociado
- ***Mecanismos Generales:*** Ofrecen comentarios extra, información, o semántica sobre un elemento de modelo. Ofrecen también los mecanismos de extensión a UML

El apartado tres se dedica a estudiar los diagramas de estructura en general, aunque el énfasis se pone en el estudio de los diagramas de clase en particular, con un nivel de detalle tal que no se entra en disquisiciones avanzadas.

Los diagramas estáticos de estructura muestran la estructura estática del modelo; en concreto muestran las cosas que existen como son las clases y tipos, su estructura interna, y su relación con el resto de elementos [OMG, 1999].

Se distinguen dos tipos diferentes de diagramas de estructura, *los diagramas de clases* y *los diagramas de objetos*. Los diagramas de clase son grafos bidimensionales que muestran los elementos modelados, conteniendo clases, paquetes, e instancias como puedan ser objetos y enlaces. Por su parte, un diagrama de objetos Es un grafo de instancias. Un *diagrama de objetos estático*, es una instancia del diagrama de clases, mostrándose como una instantánea del estado del sistema en un momento dado.

De los diagramas de estructura el más utilizado es el diagrama de clases. Como elementos propios de este diagrama se introducen los siguientes conceptos de modelado: *clases, clases parametrizadas, atributos, métodos, asociaciones binarias, asociaciones n-arias, asociaciones cualificadas, agregaciones, composiciones* y *generalización/especialización*.

Definidos estos conceptos se explican los siguientes modelados para el sistema:

- **Modelado del vocabulario**. Consta de tres pasos:
  - Identificar que es lo que emplean los usuarios o informáticos para describir el problema y conseguir su abstracción.
  - Por cada abstracción, identificar un conjunto de responsabilidades y asegurar que éstas están bien distribuidas entre las clases.
  - Añadir los atributos y operaciones que se precisan para insertar tales responsabilidades en las clases.
- **Modelado de la distribución de las responsabilidades en un sistema**. Este modelado requiere:
  - Identificar el conjunto de clases que han de trabajar juntas para materializar algún comportamiento.

- Identificar el conjunto de responsabilidades por cada una de estas clases.
- Revisar aquellas clases que asumen demasiadas responsabilidades en pequeñas abstracciones y dividirlas. Así mismo, eliminar responsabilidades triviales de las grandes abstracciones. Reubicar todas las responsabilidades, hasta conseguir un reparto razonable.
- Considerar la forma en la que unas clases colaboran con otras y redistribuir las responsabilidades en función de si alguna hace demasiado, o no hace casi nada, en el marco de la colaboración descrita.

- **El modelado de elementos *no software* requiere:**
  - Modelar aquello que se quiera abstraer como una clase.
  - Si en UML estos elementos se quieren diferenciar de otros, crear un nuevo bloque de construcción, empleando estereotipos, para especificar nueva semántica.
  - Si se quiere modelar algún elemento de hardware que contenga software, considerarlo como una clase nodo, de modo que se pueda expandir su estructura.
- **Existen características que pueden modelarse desde el lenguaje de programación.** Este es el caso de los tipos primitivos, como enteros, caracteres, cadenas, tipos enumerados... que puede crear el programador. En este caso se debe:
  - Modelar aquello que se abstrae como un tipo o una enumeración, usando notación de clases con el estereotipo adecuado.
  - Si es necesario especificar el rango de valores asociados con el tipo, utilizar restricciones.
- **Modelado de la herencia simple.** Requiere la siguiente pauta metodológica:
  - Dado un conjunto de clases, identificar atributos y operaciones que sean comunes para dos o más de ellas.
  - Elevar esas responsabilidades comunes, atributos y operaciones a una clase más general, y seguir construyendo la jerarquía, cuidando que no tenga demasiados niveles.
  - Especificar las características que las clases más especializadas van a heredar de las superclases, estableciendo las adecuadas relaciones.

- **Modelado de las asociaciones.** Se dan los siguientes pasos:
    - Para cada par de clases, cuando se necesite navegar de una a otra, especificar una asociación, que representa una conexión de regulación del intercambio de datos.
    - Para cada par de clases, si los objetos de una clase necesitan interactuar con los objetos de otra, parametrizando alguna operación, se detalla una asociación que representa una asociación de regulación de comportamientos.
    - Para cada clase especificar la multiplicidad en los extremos.
    - Si una clase del modelo es estructuralmente u organizacionalmente un nodo en la relación con otras clases que son partes del mismo, conviene estudiar el modelado como una agregación.

Un diagrama de clase puede utilizarse desde diferentes perspectivas [Fowler and Scott, 2000]:

- ***Conceptual:*** El diagrama de clase representa los conceptos en el dominio del problema que se está estudiando. Este modelo debe crearse con la mayor independencia posible de la implementación final del sistema.
- ***Especificación:*** El diagrama de clase refleja las interfaces de las clases, pero no su implementación. Aquí las clases aparecen más cercanas a los tipos de datos, ya que un tipo representa una interfaz que puede tener muchas implementaciones diferentes.
- ***Implementación:*** Esta vista representa las clases tal cual aparecen en el entorno de implementación.

El cuarto apartado se dedica al estudio de los diagramas de interacción, que son modelos que describen como grupos de objetos colaboran para conseguir algún fin [OMG, 1999]. Una interacción es, por tanto, un comportamiento que comprende un conjunto de mensajes intercambiados entre objetos dentro de un contexto.

En UML se definen dos tipos de diagramas de interacción, *los diagramas de secuencia* y *los diagramas de colaboración*. Ambos poseen una representatividad semántica equivalente, pero matizan respectivamente el orden y la estructura de los comportamientos.

Un diagrama de secuencias muestra las interacciones expresadas en función de secuencias de tiempo. En concreto muestra los objetos participantes y los mensajes que intercambian entre ellos a lo largo del tiempo [OMG, 1999].

Los diagramas de secuencia, semejantes a la traza de eventos de OMT, siguen los siguientes pasos:

- Indicar el contexto: sistema, subsistema, operación o clase.

- Identificar los componentes que intervienen.
- Asignar una línea de vida para cada uno de los objetos, con las salvedades que imponen los objetos que se crean o destruyen, para los que se deben de explicar los mensajes de *nacimiento* y *muerte*, empleando los estereotipos adecuados.
- Comenzando con el mensaje que inicia una interacción, anotar cada mensaje subsiguiente de arriba abajo, entre las líneas de vida, mostrando las propiedades de cada uno de ellos, así como sus parámetros, siempre que sea preciso para aclarar la semántica de la interacción.
- Si se necesita visualizar el anidado de mensajes en los momentos en los que la computación se efectúa, *adornar* cada línea de vida de cada objeto con este aspecto del control.
- Si se necesitan especificar restricciones de espacio o tiempo, *adornar* cada mensaje del sistema con la notación correspondiente a éstas.
- La inclusión de pre y post-condiciones en cada mensaje, permite especificar más formalmente el flujo de control.

Un diagrama de colaboración muestra cómo las instancias específicas de las clases trabajan juntas para conseguir un objetivo común. Implementa las asociaciones del diagrama de clases mediante el paso de mensajes de un objeto a otro [OMG, 1999].

Para modelar diagramas de colaboración se sigue la siguiente pauta:

- Determinar el contexto de la interacción, como en el diagrama de secuencia.
- Indicar los componentes que intervienen.
- Indicar las propiedades iniciales de cada objeto. Si, como consecuencia de la interacción, se produjera algún cambio, insertar un objeto duplicado en el diagrama, con los valores actualizados, y estableciendo como nombre de mensaje estereotipado, *become* o *copy*.
- Especificar los enlaces entre los objetos, como único soporte de los mensajes.
- Numerar adecuadamente los mensajes, para lo que se sugiere una notación decimal que favorezca la anidación.
- Si fuera preciso, por razones semánticas, escribir notas indicando restricciones de tiempo y espacio.
- Las pre y post-condiciones añaden, como en los diagramas de secuencia, la formalidad pertinente a las restricciones.

El quinto y último apartado del tema se dedica a los *casos de uso*. Los casos de uso, introducidos inicialmente por **Ivar Jacobson**, se pueden utilizar para la identificación y

documentación de los requisitos funcionales de un sistema software, así como para el modelado de las interacciones entre el sistema y los actores en los escenarios identificados.

Un caso de uso es una descripción de un conjunto de secuencias de acciones que ejecuta el sistema, y que producen un resultado útil para el actor productor del evento que arranca el citado caso de uso.

Es interesante, entre los flujos de eventos que relacionan los actores con los casos de uso, separar los normales de los calificados como excepcionales, que provocan una remodelación con nuevos casos de uso.

Cada secuencia de un caso es llamada un escenario, entendido como una secuencia típica de acciones que ilustran el comportamiento. Esto significa que un escenario, es básicamente, una instancia de un caso de uso.

Es posible utilizar los casos de uso como medio para identificar las clases cuando se muestran como entidades colaboradoras de un caso de uso, tratando en ese momento de ir asignándoles sus propiedades estructurales y de comportamiento. Estas clases se complementarán en las siguientes fases de iteración, o al definir otros casos de uso.

Para organizar los casos de uso, éstos se organizan en paquetes de la misma forma que se hace con las clases.

Los casos de uso forman sistemas (*recuadrándolos en la notación*). Para modelar el contexto de un sistema se debe:

- Identificar los actores que componen su entorno y que es posible organizar aplicando sobre ellos relaciones de especialización y generalización.
- Para mejorar la comprensión es conveniente asignar un estereotipo a cada actor.
- Indicar, mediante una relación, los casos de uso en los que interviene cada actor.
- Utilizar relaciones de inclusión y de extensión entre los casos de uso para reutilizar las especificaciones de éstos.

Bibliografía

- ***Citada en las transparencias del tema:***

[Durán y Bernárdez, 1998] Durán, A. y Bernárdez, B. "*Norma para la Recolección de Requisitos de un Sistema Software (versión 1.1)*". Apéndice en las Actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. Murcia, Noviembre 1998.

También disponible en línea en http://www.lsi.us.es/~amador/norma/recoleccion.zip [Última vez visitado: 12/1/2000] y en http://tejo.usal.es/~fgarcia/docencia/isoftware/doc/norma.pdf [Última vez visitado: 12/1/2000]. 1998.

[Fowler and Scott, 2000] Fowler, Martin and Scott, Kendall. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd edition. Addison Wesley, 2000.

[García y Pardo, 1998] García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Jacobson, 1987] Jacobson, Ivar. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I. and Booch, G. "*The Unified Modeling Language Reference Manual*". Addison-Wesley, 1999.

- ***Lecturas complementarias:***

Bell, Alex E. and Schmidt, Ryan W. "*UMLoquent Expression of AWACS Software Design*". Communications of the ACM, 42(10):55-61. October, 1999.

Informe sobre la utilización de UML en un proyecto real.

Booch. Graddy. "*Quality Software and the UML*". Object Magazine. March, 1997.

Artículo en el que Booch pone de manifiesto la necesidad de una unificación en los métodos de desarrollo como incidente en la calidad del software desarrollado.

Conallen, Jim. "*Modeling Web Application Architectures with UML*". Communications of the ACM, 42(10):63-70. October, 1999.

Utilización de UML para la especificación de aplicaciones basadas en la web.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

Artículo que da una visión histórica del nacimiento y evolución de UML hasta sus versión 1.1, presentando los objetivos que se perseguían con este lenguaje de modelado.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Diagramas de Clase en UML 1.1*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(3):71-76. Marzo, 1998.

Artículo presenta los conceptos más relevante de un diagrama de clase en UML 1.1.

Kobryn, Cris. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October, 1999.

Artículo donde se detalla el presente y el futuro de UML en su camino hacia la estandarización, ya no por OMG sino por ISO.

Jacobson, Ivar. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

Artículo en el que se dan a conocer los casos de usos internacionalmente.

Kruchten, Philippe. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November, 1995.

Artículo importante para comprender la utilización de UML dentro de un proceso software. La arquitectura software que utiliza cinco vistas concurrentes, cada una de las cuales se ocupa de un conjunto específico de conceptos.

Ohnjec, Viktor. "*Converging on OOAD Agreement*". Applications Development Trends, 4(2). February, 1997.

Artículo que recoge los esfuerzos por la unificación de los métodos de ADOO, que se derivaron de la OMG's OOAD Task Force, de la que saldría como resultado la adopción de UML como lenguaje de modelado estándar.

Whitlock, David. "*The Unified Modeling Language*". http://watson2.cs.binghamton.edu/~dwhitloc/uml/paper/part1.html. [Última vez visitado, 2-2-1999]. 1999.

Introducción a UML.

- ***Referencias utilizadas para preparar las clases:***

a) Referencias generales

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 29 de noviembre de 1999.

Definición de las tareas a realizar, los productos a obtener y las técnicas a emplear durante la actividad de análisis de requisitos de la fase de ingeniería de requisitos del ciclo de vida de la Ingeniería del Software, tomando como paradigma de desarrollo la Orientación a Objeto.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 18 de octubre de 1999.

Metodología para la recolección de requisitos de un sistema software. Define como único producto entregable el Documento de Requisitos del Sistema, que documenta los requisitos-C utilizando una combinación de casos de uso, plantillas para la documentación de requisitos y patrones lingüísticos propios del dominio de aplicación.

Fowler, Martín. "*A Survey of Object-Oriented Analysis and Design Techniques*". Technical-Report, 1997.

Excelente resumen sobre diferentes técnicas para el análisis y diseño orientado a objetos.

b) UML

Blaha, Michael and Premerlani, William. "*Object-Oriented Design of Database Applications*". Rose Architect, 1(2). January, 1999.

Presenta un enfoque sistemático para elegir un almacenamiento de datos para las aplicaciones que utilizan gran cantidad de datos.

Blaha, Michael and Premerlani, William. "*Implementing UML Models with Relational Databases*". Rose Architect, 1(3). April, 1999.

Este artículo muestra como implementar modelos UML con bases de datos relacionales.

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. "*El Lenguaje Unificado de Modelado*". Object Technology Series. Addison Wesley, 1999.

Referencia obligada, escrita por los principales responsables de UML. Es la traducción al español de [Booch et al., 1999].

Bruegge, Bernd and Dutoit, Allen H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Todo el libro utiliza UML para expresar su visión orientada al objeto de la Ingeniería del Software, pero en concreto su segundo capítulo, **Modeling with UML**, presenta los conceptos de modelado de este lenguaje.

Cantor, Murray R. "*Object-Oriented Project Management with UML*". Wiley & Sons, 1998.

Libro de gestión de proyectos que utiliza UML como una herramienta de organización.

Eriksson, Hans-Erik and Penker, Magnus. "*UML Toolkit*". John Wiley & Sons, 1998.

Libro muy recomendable para introducirse en UML. Su mayor "*desventaja*" es que explica UML 1.0.

Fowler, Martin and Scott, Kendall. "*UML Gota a Gota*". Addison Wesley Longman (Pearson), 1999.

Traducción de [Martin and Scott, 1997] (libro premiado por la revista **Software Development Magazine** en 1997). Un libro fácil de leer que puede utilizarse como introducción a UML y guía de referencia. No se adentra en aspectos avanzados.

Recientemente ha aparecido la segunda edición de este libro [Martín and Scott, 2000], todavía no traducida al español.

Larman, Craig. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Traducción de [Larman, 1998]. No es un libro plenamente dedicado ni a UML ni a los patrones. Es un libro de introducción al análisis y diseño orientado a objetos que usa ambos conceptos. Tiene un planteamiento en el que va recorriendo las fases de un desarrollo software desde la fase de definición a la fase de implementación, incorporando después tres secciones avanzadas, una dedicada al análisis, otra a los patrones de diseño y otra de temas especiales relacionados con la notación y el proceso.

Liberty, Jesse. "*Beginning Object-Oriented Analysis and Design with C++*". Wrox Press Ltd., 1998.

No se trata de un libro de UML, sino un de un libro de introducción al análisis y diseño orientado a objetos, que utiliza UML para sus modelos. Fácil de leer, muy ligado a C++, especialmente al hablar de diseño.

López, Natalie, Migueis, Jorge y Pichon, Emmanuel. "*Integrar UML en los Proyectos*". Gestión 2000, 1998.

Libro que acaba dando una visión panorámica de UML 1.0 y 1.1.

Lunn, Ken. "*Object Oriented Analysis and Design – Course Notes*". 1997.

Notas de un curso de ADOO que utiliza UML como lenguaje de modelado.

Muller, Pierre-Alain. "*Modelado de Objetos con UML*". Ediciones Gestión 2000, 1997.

Libro que, utilizando UML como base, aborda el modelado de sistemas software. En lo tocante al presente tema el capítulo 3, **La notación UML**, es el más relacionado.

Oestereich, Bernd. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

El libro al completo es una buena referencia para la explicación de UML, pero cabe destacar especialmente la parte II, **Example**, donde se desarrolla un ejemplo en los capítulos 4 y 5, y la parte III, **Fundamentals of the Unified Modeling Language**.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Documentación oficial de la versión UML 1.3. Consta de sumario, semántica, guía de notación, perfiles estándares, interfaz CORBA, especificación DTD XMI de UML, OCL, elementos estandarizados y glosario.

Pooley, Rob and Stevens, Perdita. "*Using UML Software Engineering with Objects and Components*". Addison-Wesley, 1999.

Libro muy recomendable y que se ajusta muy bien a los temas 6 y 7 de esta unidad docente. En relación con UML destacar su parte II, **The Unified Modeling Language**, donde hace un estudio de UML donde los capítulos se organizan por modelos, de forma que para los principales modelos hay un capítulo de conceptos básicos al que le sigue otro capítulo de conceptos avanzados, y su parte III, **Case studies**, donde desarrolla tres ejemplos. Mencionar también que su parte IV, **Towards practice**, trata tres temas de lo más interesante como son *la reutilización del software, la calidad del producto* y *la calidad del proceso*.

Rational Software Corporation. "*Inside the Unified Modeling Language - UML*". Multimedia Educational Tool. April, 1999.

Herramienta multimedia que contiene un tutorial sobre UML, una demostración de Rational Rose, varios modelos de ejemplo y una versión alfa de la versión 1.3 de UML.

Rumbaugh, James. "*The View from the Front: Changes to UML by the Revision Task Force*". Rose Architect, 1(3). Summer, 1999.

Cambios sufridos por UML de la versión 1.1 a la 1.3.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

Uno de los mejores libros de consulta. Organizado como si fuera un diccionario de términos, se accede rápidamente al término a consultar y se obtiene una explicación bastante completa.

c) Casos de uso

Berard, Edward V. "*Be Careful with 'Use Cases'*". The Object Agency, Inc., 1995.

Llamamiento de atención sobre la forma de utilizar los casos de uso.

Cockburn, Alistair. "*Writing Effective Use Cases*". To be published by Addison Wesley Longman in 2000. Draft 2 available at http://members.aol.com/humansandt/crystal/usecasetechnique/getWEUCbook.htm. [Última vez visitado, 14-3-2000]. December, 1999.

Una buena referencia para introducirse en los casos de uso y en su utilización en los proyectos software.

Collins-Cope, Mark. "*The Requirements/Service/Interface (RSI) Approach to Use Case Analysis (A Pattern for Structured Use Case Development)*". Ratio Group Ltd. http://www.ratio.co.uk/RSI.htm. [Última vez visitado, 19-5-1999]. 1999.

Utilización de los casos de uso para la captura de requisitos. Esta propuesta distingue tres categorías de casos e uso: *requisitos, servicios* e *interfaz*.

Christerson, Magnus. "*From Use Cases to Components*". Rose Architect, 1(1). October, 1998.

Diseño de arquitecturas basadas en componentes desde casos e uso.

Díaz, Oscar and Rodríguez, Juan José. "*Change Case Analysis*". Journal of Object-Oriented Programming (JOOP), 12(9):9-15,48. February, 2000.

Presentan un *framework* de cuatro capas para efectuar el proceso de elicitación de los casos de uso.

Fowler, Martin. "*Use and Abuse Cases*". Distributed Computing. April, 1998.

Malos usos de los casos de uso.

Iturrioz Sánchez, Juan Ignacio. "*Una Metodología para el Desarrollo de Sistemas de Base de Datos Objeto-Relacional*". Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos. Universidad del País Vasco – Euskal Herriko Unibertsitatea. Junio, 1998.

En su segundo capítulo, **Análisis de requisitos**, hace una presentación exhaustiva de los casos de uso.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Del creador de los casos de uso. Presenta su enfoque del proceso de desarrollo de software orientado a objetos, totalmente basado en los casos de uso.

Jacobson, Ivar, Griss, Martin and Jonsson, Patrik. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley, 1997.

Aunque es un libro de reutilización, los casos de uso y su reutilización, están presentes a lo largo de toda la obra. En relación con el presente tema, y más en concreto con los casos de uso, se recomienda la lectura del capítulo 3, **Object-Oriented Software Engineering**, y del capítulo 5, **Use case components**.

Kenworthy, Edward. "*Use Case Modelling. Capturing User Requirements*". http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.html. [Última vez visitado, 2-2-1999]. December, 1997.

Utilización de los casos de uso para la captura de requisitos.

Rosenberg, Doug and Scott, Kendall. "*Use Case Driven Object Modeling with UML. A Practical Approach*". Object Technology Series. Addison-Wesley, 1999.

Libro que presenta la aproximación metodológica **ICONIX Unified Object Modeling**, basada en UML y casos de uso.

Rumbaugh, James. "*Getting Started. Using Use Cases To Capture Requirements*". Journal of Object-Oriented Programming (JOOP), 7(5):8-12, 23. September, 1994.

Artículo que aborda la utilización de los casos de uso en el contexto de OMT.

Schneider, Geri and Winters, Jason P. "*Applying Use Cases. A Practical Guide*". Object Technology Series. Addison-Wesley, 1998.

Libro centrado en la construcción de casos de uso. No muy extenso y fácil de leer. Puede convertirse en el libro de referencia de un capítulo monográfico sobre casos de uso.

Texel, Putnam P. and Williams, Charles B. "*Use Cases Combined with Booch, OMT UML. Process and Products*". Prentice Hall, 1997.

Libro que introduce la utilización de los casos de usos con otras metodologías como pueden ser OMT o el método de Booch. Está escrito en plan *recetario* siendo útil como referencia ocasional y fuente de ejemplos. Como se puede deducir por su fecha de edición, es compatible con UML 1.0.

Vadaparty, Kumar. "*Use Cases - Basics*". Journal of Object-Oriented Programming (JOOP), 12(9). February, 2000.

Conceptos básico de los casos de uso.

Zhang, David D. "*Use Case Modeling for Real-time Application*". In Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'99. (27 - 29 January, 1999, Santa Barbara, California – USA). Pages 56-64. IEEE Computer Society, 1999.

Aplicación de los casos de uso en los sistemas software de tiempo real.

d) Aspectos de modelado avanzado

Odell, James J. "*Advanced Object-Oriented Analysis and Design Using UML*". Cambridge University Press. SIGS Books, 1998.

Recopilación de artículos de este autor sobre aspectos avanzados de modelado orientado a objetos, actualizados para ser congruentes con la notación de UML.

Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

No es un libro en el que UML esté presente, se dedica a la presentación del método OOram, pero sin embargo la utilización del concepto de rol como base para la realización de los modelos lo convierte en una referencia obligada en el modelado de sistemas orientados a objetos.

Rumbaugh, James. "*Disinherited! Examples of Misuse of Inheritance*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. February, 1993.

Recoge ejemplos de malos usos de la herencia.

Rumbaugh, James. "*Building Boxes: Subsystems*". Journal of Object-Oriented Programming, 7(6): 16-21. October, 1994.

Artículo sobre la organización de grandes sistemas software; introducción del concepto de subsistema.

Rumbaugh, James. "*Driving to a Solution. Reification and the Art of System Design*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. July, 1995.

Artículo que trata de utilizar la reificación para convertir algoritmos en objetos.

Rumbaugh, James. "*Taking Things in Context. Using Composites to Build Models*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. November, 1995.

Diferencias entre la composición y la agregación. También en [Rumbaugh, 1994b]

Warmer, Jos and Kleppe, Anneke. "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley. Object Technoloy Series, 1999.

Este libro explica, de una forma más detallada que en [OMG, 1999], la utilización de OCL (lenguaje de especificación de restricciones basado en la lógica de primer orden) para incluir restricciones y reglas en UML, con el fin de conseguir unos modelos más precisos.

Zhao, Liping and Foster, Ted. "*Modeling Roles with Cascade*". IEEE Software, 16(5):86-93. September/October, 1999.

La naturaleza dinámica de los objetos puede modelarse con sus roles. Este artículo propone utilizar el patrón *cascade* para representar sus roles teniendo en cuenta las relaciones entre ellos.

e) UML en aplicaciones de tiempo real

Selic, Bran. "*Turning Clockwise: Using UML in the Real-Time Domain*". Communications of the ACM, 42(10):46-54. October, 1999.

Utilización de UML para la especificación de sistemas software de tiempo real.

Rational Software Corporation. "*Unified Modeling Language for Real-Time Systems Design*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 16/6/97]. 1997.

Repaso a las técnicas de UML que mejor se ajustan para la especificación de sistemas software de tiempo real.

Selic, Bran and Rumbaugh, James. "*Using UML for Modeling Complex Real-Time Systems*". Technical Report. March, 1998.

Extensiones de UML para el modelado de software de tiempo real.

f) Otras propuestas coetáneas a UML

Firesmith, Donald G. and Henderson-Sellers, Brian. "*Upgrading OML to Version 1.1: Part 1 Referencial Relationships*". Journal of Object-Oriented Programming (JOOP), 11(3):48-57. June, 1998.

Firesmith, Donald G. and Henderson-Sellers, Brian. "*Upgrading OML to Version 1.1: Part 2 Additional Concepts and Notation*". Journal of Object-Oriented Programming (JOOP), 11(5):61-67. September, 1998.

Descripción de las mejoras introducidas en el OPEN Modeling Language (OML).

Firesmith, Donald, Henderson-Sellers, Brian, Graham, Ian and Page-Jones, Meilir. "*OPEN Modeling Language (OML) Reference Manual*". Version 1.0. OPEN Consortium. December, 1996.

Lenguaje de modelado OML.

Henderson-Sellers, Brian. "*Choosing between UML and OPEN*". 1997.

Comparativa entre UML y OPEN. Debe tenerse en cuenta que la fecha de la comparativa es 1997, con la versión 1.0 de UML todavía no publicada.

Henderson-Sellers, Brian. "*OML: Proposals to Enhance UML*". In Proceedings of *<<UML>>'98 Beyond the Notation* Conference. (3rd-4th June 98, Mulhouse - France). June, 1998.

Propuestas para corregir las deficiencias de UML.

Henderson-Sellers, Brian, **Simons, Tony and Younessi, Houman**. "*The OPEN Toolbox of Techniques*". Open Series. Addison-Wesley, 1998.

Directorio de técnicas orientadas a objetos.

Rivas, Erick, DeSilva, Dilhar, McDaniel, Terrie and Atkinson, Colin. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January, 1997.

Propuesta de Platinum Technology, Inc. a la petición de propuesta de OMG, en relación con la definición de un metamodelo para representar la semántica del ADOO. Lo más interesante de este documento es lo bien explicado que está el framework de modelos compuesto por los cuatro niveles de abstracción: *meta-metamodelo, metamodelo, modelo* y *datos*.

Tema 8: ***Visión general de la metodología OMT***

Descriptores

OMT, análisis, modelos del mundo real, requisitos, definición del problema, diseño, arquitectura del sistema, subsistemas.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Integrar las distintas técnicas aprendidas dentro de una aproximación metodológica como puede ser OMT, aunque la metodología concreta puede variar en próximos cursos, siendo sustituida por ejemplo por RUP [Jacobson et al., 1999].
- Recalcar la importancia que tiene para el éxito de un proyecto software partir de una adecuada definición del problema y utilizar un enfoque sistemático para el desarrollo de su solución.

Contenidos

8.1 Introducción
8.2 Fases
8.3 Análisis
8.4 Diseño del sistema
8.5 Diseño de los objetos

Tabla 5.15. Contenidos del tema 8 del programa de teoría de Ingeniería del Software

Resumen

El tema se organiza en cinco apartados. El primero de ellos sirve para introducir los orígenes, evolución y objetivos de la metodología OMT (*Object Modeling Technique*). Se hace especial hincapié en diferenciar entre la primera versión de OMT [Rumbaugh et al., 1991] y su evolución hacia OMT-2, menos conocida al quedar eclipsada por la eclosión de UML, que se recoge en diversos artículos de **James Rumbaugh** en la revista ***JOOP*** (*Journal of Object-Oriented Programming*) [Rumbaugh, 1995a], [Rumbaugh, 1995b], [Rumbaugh, 1995c], [Rumbaugh, 1995d] y posteriormente en su libro ***OMT Insights*** [Rumbaugh, 1996].

OMT-2 puede considerarse como la precursora de la versión 0.8 del método unificado de **Booch y Rumbaugh** [Booch and Rumbaugh, 1995].

En el apartado dos se enuncian y se presentan de forma somera las cuatro fases de que consta esta metodología: *análisis, diseño del sistema, diseño de los objetos* e *implementación*.

El tercer apartado se dedica a la fase de análisis, el punto más sobresaliente de esta metodología. Dicha fase consta de cinco actividades fundamentales:

1. *Definición del problema.*
2. *Construcción del modelo de objetos.*
3. *Desarrollar el modelo dinámico.*
4. *Generar el modelo funcional.*
5. *Verificar, iterar y refinar los tres modelos.*

Se explica la importancia que tiene la obtención de los requisitos en la etapa de definición del problema. En el proceso de análisis se utilizan como entradas, además de las especificaciones del problema, las entrevistas mantenidas con los usuarios, el conocimiento personal de los ingenieros del software sobre el dominio del problema y la experiencia que se haya adquirido sobre el mundo real ante planteamientos parecidos.

Para la construcción de un modelo de objetos, que va a representar la estructura estática de los datos que se corresponden con el mundo real que se está estudiando, se indica lo importante de construir inicialmente el modelo al más alto nivel de abstracción, incluyendo las asociaciones entre las clases. Se deben posponer a etapas ulteriores los refinamientos de la inclusión de generalización y especialización e incluso algunas estructuras de agregación.

La determinación de los atributos y métodos correspondientes a las clases puede efectuarse en una iteración posterior del modelo. Debe transmitirse al alumno que no debe obsesionarse en descubrirlos todos desde el primer momento. Conviene aclarar que no todos los atributos encontrados son independientes y es su obligación eliminar, o al menos marcar como tales, a los atributos derivados.

Después de obtener el primer modelo surge la necesidad de refinarlo. Se incluirán en estos refinamientos las relaciones de generalización/especialización, las cardinalidades de las asociaciones y la vías de acceso. Posteriormente se podrá estudiar la inclusión de agregaciones complejas y otras estructuras.

El modelo de objetos final difícilmente se alcanzará en una sola aproximación, siendo necesario realizar varias iteraciones. Inclusive alguna de estas iteraciones se realizará después de realizar los otros modelos propios de la fase de análisis.

Una vez conseguido el modelo estático suficientemente descriptivo, y en el que las asociaciones representen el sentido del sistema, el paso siguiente es lograr un modelo dinámico, que regule el flujo de control de los comportamientos. Para este fin debe construirse primero un escenario. Una manera adecuada para ello consiste en plantear el

típico diálogo entre el sistema y el usuario. De esta forma se tendrá una idea del comportamiento que se espera del sistema. Conviene insistir en la necesidad de construir no sólo escenarios normales, sino también especiales o correspondientes a situaciones excepcionales. Se propone construir uno o varios escenarios de *éxito*, alguno de *fracaso* y alguno de *bloqueo*.

En la mayor parte de los sistemas en los que la interactuación con el usuario juega un papel primordial, la interactuación se puede dividir en dos partes: *lógica de aplicación* e *interfaz de usuario*. El análisis debe concentrarse en el flujo de datos, dejando en segundo plano el formato de las ventanas, actividad que se corresponde con el diseño. Lo importante es destacar que el modelo dinámico explica la lógica de control de la aplicación.

Después de depurar los escenarios, el siguiente paso consiste en la construcción de la traza de eventos a partir de los escenarios, procediendo a continuación a crear los diagramas de transición de estados de cado uno de los objetos de la traza que interactúa y cambia de estado. Puede añadirse, no obstante, una etapa intermedia para obtener el *diagrama de flujo de eventos*.

Cada diagrama de transición de estados corresponde a una sola clase, dando lugar la relación entre ellos al modelo dinámico, logrando la interacción de las clases mediante el envío de mensajes y/o las solicitudes de los servicios.

El modelo funcional, sirve para identificar los valores de entrada y salida, sin tener en cuenta la secuencia de los procesos, las decisiones de control ni las estructuras de los objetos.

Utiliza DFDs para representar las dependencias funcionales. Los pasos que marca OMT son [Rumbaugh et al., 1991]: *identificación de los valores de entrada y salida, construir los diagramas de flujo de datos que determinan las dependencias funcionales, describir las funciones, identificar las restricciones y especificar los criterios de optimización*.

El modelo funcional ha sido bastante criticado, porque puede llevar a concebir una descomposición funcional del sistema en lugar de seguir un enfoque orientado al objeto. Actualmente [Rumbaugh, 1994a], se propone que este modelo consista en la descripción detallada de las operaciones del sistema, utilizando plantillas, y sólo en el caso en que una operación afecte a varios objetos del sistema, se dibujará el diagrama de flujo de datos orientado a objetos.

El cuarto apartado se centra en el diseño del sistema. En la fase de análisis lo fundamental es determinar lo que debe hacerse, sin que sea necesario, ni conveniente comprometerse sobre la forma en que se hará. Sin embargo, durante el diseño es necesario tomar decisiones sobre la forma en que se resolverá el problema. El diseño del sistema viene a representar la primera aproximación a este respecto.

La primera tarea que se lleva a cabo es la descomposición del sistema en subsistemas. Un subsistema se puede definir como *el conjunto de componentes del sistema de mayor nivel que tienen propiedades o funcionalidades similares, se encuentran ubicados en el mismo lugar o se ejecutan con el mismo hardware en una temporalidad continuada y congruente con la secuencia de resultados*.

Desde el punto de vista de la orientación a objetos, un subsistema puede entenderse como un conjunto de clases, asociaciones, operaciones, sucesos y restricciones interrelacionados que posee, además, una interfaz bien definida y concisa con los demás subsistemas.

En OMT el sistema se divide en subsistemas como un conjunto de niveles o capas horizontales y de particiones verticales.

La segunda etapa es la identificación de la concurrencia. Aunque en el análisis se considere que todos los objetos son concurrentes, a la hora de implementar un sistema, un procesador puede soportar varios objetos, por lo que es necesario identificar qué objetos pueden actuar concurrentemente y cuáles no.

La tercera etapa sería la de ubicar los subsistemas en procesadores y tareas. En esta etapa el diseñador debe estimar los recursos de hardware y software necesarios para implementar el sistema, ubicando los subsistemas en distintos procesadores.

La cuarta etapa conlleva la gestión de los almacenes de datos. Conviene indicar en primer lugar que éstos facilitan la separación de sistemas, ya que permiten por una parte que un subsistema realice la captura y representación de la información, mientras que otro se encarga de las tareas de inserción, actualización y modificación de datos.

La quinta etapa es determinar los mecanismos de control de acceso a recursos globales.

La sexta etapa conlleva la elección del enfoque para la implementación del control. Las interacciones entre los objetos reflejadas en los diagramas de transición de estados, que en su conjunto configuran el modelo dinámico, tienen que implementarse. Se admiten, normalmente, la existencia de dos flujos de control: e*l interno* y el *externo*, siendo este último el que se corresponde con los sucesos, es decir, eventos que se reciben desde los agentes externos al sistema.

El control de los sucesos internos puede hacerse o secuencialmente, controlado por procedimientos, o bien por eventos. Otra posibilidad es que se hiciese concurrentemente.

El control interno se corresponde con el flujo de control dentro de un proceso, y está íntimamente relacionado con la implementación.

La séptima etapa lleva a considerar las condiciones de entorno, especialmente la iniciación y la finalización, así como la aparición de posibles fallos.

En la octava y última etapa se establecen las prioridades entre los distintos objetivos de diseño: recursos de memoria, tiempo de respuesta, portabilidad...

El quinto y último apartado está dedicado al diseño de los objetos, que es donde se pasa de la orientación del mundo real propia del modelo de análisis a una orientación más propia del mundo de los ordenadores requerida para llevar a cabo una implementación práctica.

En esta fase se lleva a cabo la estrategia de diseño del sistema y se completan los detalles. También se añaden nuevos objetos, de modo que los objetos del dominio del problema ya no sean los únicos, puesto que aparecen nuevos objetos para lograr la "*implementación*" ya dentro del dominio de la solución.

Así, se pueden resumir las actividades a realizar en esta fase en los siguientes puntos:

- Obtener las operaciones de las clases a partir de otros modelos. Así, por ejemplo, un cambio de estado en un objeto puede corresponder a una operación.
- Diseñar algoritmos para implementar las operaciones, definiendo clases y operaciones si fuera necesario.
- Optimizar los caminos de acceso a los datos, almacenando atributos o interrelaciones redundantes. Aquí se incluye el acceso a los datos persistentes, almacenados típicamente en bases de datos.
- Implementar el control seleccionado en la fase anterior, identificando los caminos principales y alternativos.
- Ajustar la estructura de clases para incrementar la herencia. La herencia puede facilitar bastante el proceso de creación de código en el momento de la implementación, pero ha de tenerse en cuenta en cuenta que se deriva de la relación de generalización/especialización y ésta se utiliza en el análisis principalmente con otros fines. Ajustar la estructura de clases, permitiendo la reutilización de código, es un principio básico de la programación orientada a objetos. Se trata de abstraer el comportamiento y la estructura común de las clases, y según que casos, justificar la creación de una superclase, abstracta en muchos casos, y varias subclases.
- Diseñar las asociaciones. Una asociación puede implementarse mediante colecciones de punteros, o de referencias, en las clases que asocia, o mediante un nuevo objeto que contuviera los punteros, o referencias, correspondientes a los objetos de ambas clases.
- Determinar la representación exacta de los objetos, decidiendo los tipos para los atributos, si se combinan grupos de objetos en uno solo...

- Empaquetar las clases y asociaciones en módulos, ocultando en la parte privada de éstos toda la información que resulte conveniente.

Bibliografía

- ***Citada en las transparencias del tema:***

[Rumbaugh, 1994] Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

[Rumbaugh, 1996] Rumbaugh, James. "*OMT Insights*". SIGS Books Publications, 1996.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1998] Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

- ***Lecturas complementarias:***

Amako, Katsuya. "*Object Modeling Technique - Summary Note*". http://arkhp1.kek.jp/managers/computing/activities/OO_CollectInfor/Methodologies/OMT/OMTBook/OMTBook.html. [Última vez visitado, 9-2-2000]. June, 1995.

Esquema de la metodología OMT.

Brinkkemper, Sjaak, Hong, Shuguang, Bulthuis, Arjan and van den Goor, Geert. "*Object-Oriented Analysis and Design Methods a Comparative Review*". http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html. [Última vez visitado, 9-2-2000]. January, 1995.

Resúmenes de diferentes métodos orientados a objetos de primera y segunda generación, entre los que se encuentra OMT. Finalmente se presenta una comparativa de ellos.

ICON Computing, Inc. "*A Comparison of OOA & OOD Methods*". Icon Computing Inc. http://www.iconcomp.com/papers/comp/. [Última vez visitado, 9-2-2000]. 1995.

Resúmenes de diferentes métodos orientados a objetos de primera y segunda generación, entre los que se encuentra OMT. Incluye una tabla comparativa de los métodos estudiados.

Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

James Rumbaugh reconoce que el modelo funcional es el punto más controvertido de OMT, tal y como se describe esta metodología en [Rumbaugh et al., 1991], para lo que se propone la utilización de DFDs, que, en opinión del propio Rumbaugh, siempre fueron poco entendidos y nada aceptados. Así, propone un nuevo modelo funcional compuesto de casos de uso y descripciones de operaciones y de diagramas de interacción de objetos.

Rumbaugh, James. "*The OMT Process*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. May, 1994.

Aquí **Rumbaugh** explica el proceso de desarrollo que se encuentra detrás de OMT, basado en un ciclo de vida iterativo e incremental, con una naturaleza fractal difícil de explicar con los ciclos de vida lineales tipo cascada.

Rumbaugh, James. "*OMT: The Object Model*". Journal of Object-Oriented Programming, 7(8):21-27. January, 1995.

Repaso al modelo objeto de OMT.

Rumbaugh, James. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming, 7(9):6-12. February, 1995.

Repaso al modelo dinámico de OMT.

Rumbaugh, James. "*What Is a Method?*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. October, 1995.

En este artículo **Rumbaugh** explica lo qué es y de qué está compuesto un método de desarrollo. Uno de los aspectos más interesantes de este artículo es como introduce los patrones de diseño en el proceso de desarrollo del software, como un mecanismo para reutilizar la experiencia en el diseño de sistemas software.

- ***Referencias utilizadas para preparar las clases:***

a) OMT

Blaha, Michael and Premerlani, William. "*Object-Oriented Modeling and Design for Database Applications*". Prentice-Hall, 1998.

Estos dos co-autores de OMT dedican este libro a la utilización de OMT para la creación de aplicaciones de bases de datos. Utilizan notación OMT para expresar sus modelos.

Karma, Gerald M. and Casselman, Ronald S. "*A Cataloging Framework for Software Development Methods*". IEEE Computer, 26(2):34-45. February, 1993.

Artículo que presenta un marco de trabajo para catalogar métodos de desarrollo de software, aplicándolo a OMT.

Piattini, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Dentro de su capítulo 10, **Análisis y diseño orientado al objeto**, dedican el apartado 10.4 a hacer un resumen de la metodología OMT.

Premerlani, William J., Blaha, Michael R., Rumbaugh, James E. and Varwing, Thomas A. "*An Object-Oriented Relational Database*". Communications of the ACM, 33(11):99-109. November, 1990.

Combinación de un gestor de base de datos relaciones y lenguajes de programación OO en aplicaciones de gestión, utilizando OMT como método de desarrollo.

Rumbaugh, James. "*Getting Started. Using Use Cases To Capture Requirements*". Journal of Object-Oriented Programming (JOOP), 7(5):8-12, 23. September, 1994.

Artículo que aborda la utilización de los casos de uso en el contexto de OMT.

Rumbaugh, James. "*A Private Workspace. Why a Shared Repository Is Bad for Large Projects*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. September, 1995.

Rumbaugh ataca los repositorios centrales compartidos en los desarrollos donde un equipo trabaja en paralelo. Defiende que cada desarrollador debe contar con su espacio de trabajo privado en el que pueda trabajar con una versión estable del sistema, hasta que los cambios estén listos para ser compartidos con el resto del equipo.

Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books, 1996.

Recopilación de artículos de **James Rumbaugh** publicados en la revista *Journal of Object-Oriented Programming* (JOOP). Muestran la evolución de las ideas de OMT hacia OMT-2, especialmente la sección denominada **OMT Summary**.

Rumbaugh, James, Blaha, **Michael, Premerlani, William, Eddy, Frederick, Lorensen, William**. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Es la traducción al español del libro clásico sobre OMT [Rumbaugh et al., 1991], libro que seguirá siendo una referencia válida aunque OMT quede desfasada, porque ha sido y es base para el desarrollo de nuevas metodologías orientadas a objetos y porque su presentación del modelo objeto y de la fase de análisis es excepcional. La obra que se referencia aquí es la segunda reimpresión que incluye unos apéndices escritos por **Luis Joyanes Aguilar** sobre OMT-2 y UML. Salvo estos apéndices, sigue siendo totalmente válida la primera edición en español que data de 1996.

b) RUP[24]

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

La guía completa del proceso software RUP.

[24] Aunque el centro de este tema es OMT, en futuros cursos puede ampliarse este tema o cambiarse radicalmente para el estudio de RUP; de ahí que se contemplen aquí las referencias con que se cuenta para su estudio. Además, su lectura puede serle útil al docente para comparar una metodología consolidada como OMT, con las nuevas tendencias metodológicas.

Kruchten, Philippe. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November, 1995.

Artículo importante para comprender la utilización de UML dentro de un proceso software. La arquitectura software que utiliza cinco vistas concurrentes, cada una de las cuales se ocupa de un conjunto específico de conceptos.

Kruchten, Philippe. "*A Rational Development Process*". Crosstalk, 9(7):11-16. July, 1996.

En este artículo el Dr. Kruchten, director de Proceso de Desarrollo en Rational Software Corporation, expone las bases de un proceso de desarrollo software evolutivo e incremental, que posteriormente se convertiría en el RUP (Rational Unified Process).

Kruchten, Philippe. "*A Rational Development Process*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 4-2-1999]. 1996.

Resumen del Rational Objectory Process, el antecesor de RUP.

Kruchten, Philippe. "*The Rational Unified Process. An Introduction*". Object Technology. Series Addison-Wesley, 1999.

En este libro se ofrece una introducción al RUP que, más que ser completa, pretende asentar los principios básicos sobre los que se basa este proceso de desarrollo. Se divide en dos partes, en la primera de ellas, **The process**, se describe el proceso, su contexto, su historia, su estructura y su ciclo de vida de desarrollo de software. En la segunda parte, **Process Workflows**, se da un repaso a varios componentes del proceso o *workflows*.

Rational Software Corporation. "*A Rational Approach to Software Development Using Rational Rose 4.0*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 1/12/98]. 1998.

Soporte de RUP con herramientas CASE (Rational Rose) y su relación con UML.

Rational Software Corporation. "*Rational Unified Process. Best Practices for Software Development Teams*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 1/12/98]. 1998.

Introducción a RUP.

Rational Software Corporation. "*Reaching CMM Levels 2 and 3 with the Rational Unified Process*". Rational Software Corporation. http://www.rational.com. [Última vez visitado, 12-2-2000]. 1998.

Cómo conseguir los niveles de 2 y 3 del modelo CMM utilizando RUP.

Rational Software Corporation. "*Rational Unified Process. Product Information FAQ*". Rational Software Corporation. http://www.rational.com. [Última vez visitado, 8-2-2000]. 2000.

Respuestas a las pregunta más frecuentes sobre RUP.

Rational Software Corporation and Context Integration. "*Building Web Solution with the Rational Unified Process: Unifying the Creative Design Process and the Software Engineering Process*". White Paper. http://www.rational.com. 1999.

Describe cómo RUP puede utilizarse para el desarrollo de aplicaciones Web.

c) Otras metodologías y métodos de desarrollo orientado a objetos

D'Souza, Desmond F. and Wills, Alan Cameron. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

Método de desarrollo basado en objetos, componentes y *frameworks* que utiliza UML como notación.

Graham, Ian, Henderson-Sellers, Brian and Younessi, Houman. "*The OPEN Process Specification*". ACM Press Books. Addison-Wesley. The OPEN Series, 1997.

Libro en el que se describe OPEN como una metodología orientada a objetos de tercera generación, que contiene un conjunto de patrones de proceso. El libro tiene dos partes. La primera de ellas, compuesta de siete capítulos, donde se describe OPEN. La segunda parte es una colección de apéndices.

Firesmith, D. and Henderson-Sellers, B. "*Improvements to the OPEN Process Metamodel*". Journal of Object-Oriented Programming (JOOP), 12(7):30-35. November/December, 1999.

Mejoras en el metamodelo del proceso de desarrollo de software de OPEN.

Henderson-Sellers, Brian. "*The OPEN-Mentor Methodology*". Object Magazine, 6(9):56-59. November, 1996. Available online at http://www.open.org.au/Publications/Documents/open.pdf [Última vez visitado, 20-3-2000].

Presentación de OPEN.

Henderson-Sellers, Brian and Graham, Ian. "*Process and Product Life Cycles: OPEN's Version 2 Life Cycle Model*". Journal of Object-Oriented Programming (JOOP), 13(1):23-26,39. March/April, 2000.

Descripción del ciclo de vida asociado a la versión 2 de OPEN.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Descripción de OOSE de **Ivar Jacobson** (también conocida como *Objectory*). Imprescindible para conocer los orígenes de RUP.

Tkach, Daniel, Fang, Walter and So, Andrew. "*Visual Modeling Technique*". Addison-Wesley, 1996.

Describe una metodología OO para el desarrollo visual de aplicaciones. Toma como base metodológica y como notación a OMT, pero la completa con casos de uso, tarjetas CRC y otras aportaciones de otras metodologías.

Unidad Docente IV: Miscelánea

Objetivo genérico

El objetivo que se plantea esta unidad docente es la somera presentación de temas adicionales que, relacionados con la Ingeniería del Software, no han tenido cabida en otras unidades docentes de mayor envergadura por motivos de tiempo.

Los temas que se traten son introducciones para que el alumno sepa de su existencia y pueda profundizar por su cuenta si lo creyese necesario. Incluso podrían sustituirse las horas reservadas a esta unidad docente por la celebración de seminarios dirigidos por el profesor y desarrollados por los alumnos sobre diferentes temas de interés, o por conferencias invitadas.

Siendo realistas, esta unidad docente es muy difícil que se llegue a impartir porque, aún cumpliendo perfectamente los tiempos programados para las otras unidades docentes, todos los cursos académicos surgen imprevistos que obligan a perder alguna clase.

Por este motivo, el 4% del temario teórico reservado para esta unidad docente, sólo lo ocupa un tema, en este caso dedicado a las herramientas CASE, por ser un tema que no debiera de faltar en el currículo de un ingeniero técnico en informática [Granger and Little, 1996].

Otros candidatos a ocupar este lugar, o a convertirse en seminarios dentro de esta asignatura, pueden ser *prueba del software, calidad del software, gestión de la configuración, mantenimiento, reutilización, reingeniería* o *aspectos legales del software.*

Tema 9: ***Herramientas CASE***

Descriptores

Tecnología CASE, repositorio, metamodelo, metadatos, ICASE, IPSE, gestión de la configuración.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **P4** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Utilización de herramientas CASE para la gestión y desarrollo de sistemas software..

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en las herramientas CASE como apoyo y automatización al proceso de desarrollo de software.
- Presentar la diferente tipología de herramientas CASE que existen en el mercado y la importancia de su integración para conseguir entornos avanzados de Ingeniería del Software.
- Indicar la importancia del intercambio de datos entre diferentes herramientas CASE para favorecer la integración, reutilización e intercambio de información entre proyectos.
- Explicar la arquitectura que se encuentra detrás de una herramienta CASE.

Contenidos

9.1 Introducción
9.2 Componentes de una herramienta CASE
9.3 Clasificación de las herramientas CASE
9.4 Integración de CASE

Tabla 5.16. Contenidos del tema 9 del programa teórico de Ingeniería del Software

Resumen

El tema se ha dividido en cuatro apartados principales, todos ellos tratados de una forma introductoria.

En el primer apartado se introduce la tecnología CASE. Se justifica la existencia de estas herramientas para aumentar la productividad y la calidad del software desarrollado, automatizando en el mayor grado posible el proceso software. Igual que la Informática, el software es un activo en prácticamente la totalidad de los dominios de aplicación o negocios actuales, debe serlo para la propia Ingeniería del Software.

Se presenta también en este primer apartado la evolución histórica de la tecnología CASE, desde sus orígenes a mediados de la década de los setenta, su eclosión a

mediados de los ochenta, su crisis a finales de los ochenta y principio de la década de los noventa, finalizando con su resurgir con mayor madurez y menos mitología a finales de los noventa.

El segundo apartado se dedica a la presentación de los componentes principales de una herramienta CASE: *repositorio, metamodelo, generador de informes, carga/descarga de datos, comprobación de errores* e *interfaz de usuario*.

En el tercer apartado se estudian diferentes clasificaciones de las herramientas CASE, propuestas por otros tantos autores.

El cuarto y último apartado es quizás el de mayor relevancia, ocupándose de los problemas de integración de las herramientas CASE.

Se mencionan las distintas formas de utilización de las herramientas CASE:

- Como herramientas aisladas, en cuyo caso sólo debe abordarse la compatibilidad con los elementos del entorno.
- En pequeños grupos, que se comunican directamente, para los que la integración está predefinida de manera propietaria.
- En presencia de un marco de integración más amplio, en cuyo caso habrá que abordar la capacidad de la herramienta para utilizar servicios relevantes del marco de integración.

Centrando la atención en los entornos ICASE (*Integrated CASE*) éstos deben cumplir ciertos requisitos, a saber:

- Proporcionar un mecanismo para compartir la información de Ingeniería del Software entre todas las herramientas contenidas en el entorno.
- Hacer posible que un cambio en un elemento de información se siga hasta los demás elementos de información relacionados.
- Proporcionar un control de versiones y una gestión de configuración general para toda la información de la Ingeniería del Software.
- Permitir un acceso directo y no secuencial a cualquiera de las herramientas contenidas en el entorno.
- Establecer un entorno automatizado para un contexto de procedimientos.
- Capacitar a los usuarios de cada una de las herramientas para experimentar una utilización consistente en la interfaz hombre-máquina.
- Permitir la comunicación entre ingenieros del software.
- Recoger métricas de gestión y técnicas para mejorar el proceso y el producto.

Deben estudiarse los diferentes niveles de integración de las herramientas CASE [Garbajosa y Bonilla, 1995]:

- ***Integración de datos:*** Mide el grado en que los datos generados por una herramienta se hacen accesibles para su uso en otras herramientas. Cabe destacar algunas iniciativas al respecto:
  - **EIA/CDIF** (*Electronic Industries Association's CASE Data Interchange Format*) [EIA CDIF Division, 1996].
  - **PCTE** (*Portable Common Tool Environment*) [Aldis, 1995].
  - **XMI (XML Metadata Interchange)** [Brodsky, 1999].
- ***Integración de control:*** Facilidad de las herramientas para comunicarse, cooperar, y la relativa sencillez con la que una nueva herramienta puede hacer uso de servicios ya ofrecidos en lugar de duplicarlos en su propio código.
- ***Integración de presentación:*** Homogeneidad y consistencia de la interfaz de usuario, por ejemplo, que en todas las herramientas se acceda a la ayuda de la misma forma, mismas barras de herramientas, mismo uso del ratón…

Por último se mencionan dos modelos de integración ya clásicos: *el modelo cebolla* y *el modelo tostadora* [Piattini y Daryanani, 1995].

Bibliografía

- ***Citada en las transparencias del tema:***

[Fuggetta, 1993] Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December, 1993.

[McClure, 1992] Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992.

[Piattini et al., 1996] Piattini, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Piattini y Daryanani, 1995] Piattini, Mario G. y Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Schmerl, 1996] Schmerl, Bradley. "*Configuration Management and Version Control*". http://tuvalu.csflinders.edu.au/seweb/scm/lectures/cmvc1.html. 23 May 1996.

[Sommerville, 1996] Sommerville, Ian. "*Software Engineering*". 5th Edition. Addison-Wesley, 1996.

- ***Lecturas complementarias:***

Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December, 1993.

Artículo que realiza un informe clasificatorio sobre la tecnología CASE.

Iivari, Juhani. "*Why Are CASE Tools Not Used*". Communications of the ACM, 39(10):94-103. October, 1996.

El autor examina la actitud hacia el uso de herramientas CASE en una amplia gama de organizaciones.

Sharma, Srinarayan and Rai, Arun. "*CASE Deployment in IS Organizations*". Communications of the ACM, 43(1):80-88. January, 2000.

Informe de la presencia de la tecnología CASE en los sistemas de información de las empresas.

- ***Referencias utilizadas para preparar las clases:***

Aldis, Margaret. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

Buena referencia para conocer más sobre el PCTE y los ICASE.

EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

Información sobre CDIF.

Fisher, Alan S. "*Tecnología CASE. Herramientas de Desarrollo de Software*". 2ª Edición. Anaya Multimedia, 1993.

Libro que sirve de introducción a la tecnología CASE, aunque quizás un tanto desfasado en cuanto a la información sobre las herramientas CASE que aborda, téngase en cuenta que es la traducción de [Fisher, 1991], de hecho la Orientación a Objetos ni se menciona.

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

En concreto el tema 6, *Ingeniería del Software Asistida por Ordenador*, donde se desarrollan los apartados que conforman el presente tema.

Long, Fred and Morris, Ed. "*An Overview of PCTE: A Basis for a Portable Common Tool Environment*". Technical Report CMU/SEI-93-TR-1, ESC-TR-93-175. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., March, 1993.

Otra referencia para profundizar en el PCTE.

Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992.

La edición inglesa de este libro data de 1989, y es todo un clásico en el área de la tecnología CASE. Presenta una panorámica muy general de todos los aspectos de esta área.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

De este libro se destaca el capítulo 19, **Análisis y diseño asistido por ordenador: CASE**, y el apéndice A, **Herramientas CASE**.

Piattini Velthuis, Mario G. y Daryanani Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

Libro que profundiza en algunos aspectos de las herramientas CASE a la vez que las engloba en un entorno más amplio, con otros aspectos que también persiguen la mejora de la calidad y productividad del desarrollo software (*métricas, modelo de proceso software, modelos de evaluación de la capacidad de desarrollo...*).

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición, McGraw Hill, 1998.

En este libro se aborda de una forma sobresaliente la introducción a la tecnología CASE en el capítulo 29, **Ingeniería del Software Asistida por Computadora**.

5.2.1.3 Bibliografía empleada en la parte teórica de la asignatura

En este apartado se va a citar todas las referencias utilizadas para impartir la parte de teoría de la asignatura de Ingeniería del Software. Para ello se van a distinguir los tres apartados que se han venido presentando en el desarrollo comentado de cada uno de los temas de este programa: *las referencias citadas en las transparencias, las lecturas complementarias* y *las referencias para preparar las clases*.

Bibliografía citada en las transparencias

Para cada uno de los temas del programa de teoría cuenta con su correspondiente juego de transparencias, que le es facilitado al alumno para seguir las clases. En estas transparencias aparecen las siguientes referencias bibliográficas:

Andreu, Rafael, Ricart, Joan y Valor Josep. "*Estrategia y Sistemas de Información*". 2ª Ed. McGraw-Hill (*serie de management*), 1996.

Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

Berard, Edward V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc., 1996.

Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, p. 20, 1992.

Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". ACM Software Engineering Notes, 11(4):22-42. 1986.

Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

Buxton, J. M., Naur, P. and Randell, B. (eds.) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

CERN. "*STING Software Engineering Glossary*". CERN. http://dxsting.cern.ch/sting/glossary.html. April 1997.

Coleman, D., Arnold, P. and Bodoff, S. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

Champeaux, Dennis, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison Wesley, 1993.

Dahl, Ole-Johan, Dijkstra, E. and Hoare, C. A. R. "*Structured Programming*". Academic Press, 1972.

Date, C. J. "*An Introduction to Database Systems*". 6th Edition, Addison-Wesley, 1995.

Dennis, J. "*Modularity*". In *Advanced Course on Software Engineering*, F. L. Bauer (editor), Springer-Verlag, pages 128-192, 1973.

DoD. "*SRI Reuse Glossary*". DOD, http://sw-eng.falls-church.va.us/ReuseIC/policy/glossary/glossary.htm, December 1995.

Durán, A. y Bernárdez, B. "*Norma para la Recolección de Requisitos de un Sistema Software (versión 1.1)*". Apéndice en las Actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. Murcia, Noviembre 1998. También disponible en línea en http://www.lsi.us.es/~amador/norma/recoleccion.zip [Última vez visitado: 12/1/2000] y en http://tejo.usal.es/~fgarcia/docencia/isoftware/doc/norma.pdf [Última vez visitado: 12/1/2000]. 1998.

Emery, J. C. "*Sistemas de Información para la Dirección. El Recurso Estratégico y Crítico*". Ed. Díaz de Santos, 1990.

Fowler, Martin and Scott, Kendall. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd edition. Addison Wesley, 2000.

Frakes, William B., Fox, Christopher, Nejmeh and Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman). Pages 2-8. IEEE Computer Society Press, 1987.

Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December, 1993.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel. *"Mecano: Una Propuesta de Componente Software Reutilizable*". En las actas de las II Jornadas de Ingeniería del Software (Donostia-San Sebastián, España, 3-5 de septiembre de 1997): 232-244. 1997.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Introducción al Análisis y Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(2): 64-70. Febrero, 1998.

Girow, Andrew. "*Objects and Binary Relations*". Object Currents, 1(6), SIGS Publications, June 1996.

Girow, Andrew. "*Binary Relations Approach to Building Object Database Model*". Object Currents, 1(11), SIGS Publications, November 1996.

Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

Hatley, Derek J. and Pirbhai, Imtiaz. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

Horan, Peter "*Software Engineering - A Field Guide***"**. Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

Houghton Mifflin Company. "*The American Heritage Dictionary of the English Language*". 3rd Edition, Houghton Mifflin Company, Electronic Version. 1992.

Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science,* Van Nostrand Reinhold, p. 1218. 1993.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 2: Process Standards*". IEEE Computer Society Press, 1999.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 4: Resource and Technique Standards*". IEEE Computer Society Press, 1999.

ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

Jacobson, Ivar. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. Osborne McGraw-Hill. 1998.

Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

Leaney, John. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. **http://www.ee.uts.edu.au/~jrleaney/setut.htm.** October 1995.

Liskov, B. "*Data Abstraction and Hierarchy*". SIGPLAN Notices, Vol. 23(5), 1988.

Maddison, R. N. "*Information System Methodologies*". Wiley Henden, 1983.

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992.

Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd edition. Prentice Hall, 1997.

Monarchi, David E. and Puhr, Gretchen I. "*A Research Typology for Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):35-47. September, 1992.

Monforte, Manfredo. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978.

National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

Parnas, David Lorge and Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4):259-265, December 1987.

Peña Marí, Ricardo. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Ed. Prentice-Hall, 1998.

Pepper, Jon. "*Object Magazine Survey: What's Corporate America Spending on Objects?*". Object Magazine, February, 1997.

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Piattini Velthuis, Mario G. y Daryanani Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

Raghavan, S., Zelesnik, G. and Ford, G. "*Lecture Notes on Requirements Elicitation*" CMU/SEI-94-EM-10, Pittsburgh (EEUU), Software Engineering Institute (Carnegie Mellon University). 1994.

Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

Royce, W. W. "*Managing the Development of Large Software Systems: Concepts and Techniques*". In Proceedings WESCON. August, 1970.

Rubin, Kenneth S. and Goldberg, Adele. "*Object Behavior Analysis*". Communications of the ACM 35 (9): 48-62. September, 1992.

Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

Rumbaugh, James. "*What Is a Method*". JOOP. October, 1995.

Rumbaugh, James. "*OMT Insights*". SIGS Books Publications, 1996.

Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Rumbaugh, J., Jacobson, I. and Booch, G. "*The Unified Modeling Language Reference Manual*". Addison-Wesley, 1999.

Schmerl, Bradley. "*Configuration Management and Version Control*". http://tuvalu.csflinders.edu.au/seweb/scm/lectures/cmvc1.html. 23 May 1996.

Shaler, S. and Mellor, S. "*Object Life Cycles: Modeling the World in States*". Prentice-Hall, 1992.

Shaw, M., and Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

Smith, M. and Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

Sommerville, Ian. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

Stevens, W.P., Myers G.J. and Constantine, L.L. "*Structured Design*". IBM Journal, 13(2):115-119, 1974.

Sutherland, Jeff. "*Object World Tutorial - Object Design Tutorial*". 1997.

Taylor, E. S. "*An Interim Report on Engineering Design*". Massachusetts Institute of Technology, 1959.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

Wasserman, A. "*Information Systems Design Methodology*". In *Software Design Techniques*. P. Freeman y A. Wasserman Editors., 4th Edition, IEEE Computer Society Press, 1983.

Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

Wolff, J. G. "*The Manegement of Risk System Development: 'Project SP' and the 'New Spiral Model'*". Software Engineering Journal, May 1989.

Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

Lecturas complementarias

Cada tema se termina con una serie de recomendaciones sobre lecturas complementarias que el alumno puede consultar para incidir más en algún aspecto del tema, o para cubrir aspectos que, en el desarrollo del propio tema, han sido mencionados de pasada.

Normalmente, se trata de artículos e informes técnicos no establecidos en la referencia básica de consulta que se le recomienda al alumno al principio del curso, y que en la mayoría de los casos utiliza el profesor para preparar *aspectos colaterales* de sus clases. Todos ellos son accesibles al alumno a través de Internet o a través del profesor. La mayor parte de ellos están es inglés, lo que a menudo es una barrera para el alumno, pero por otra parte debe hacérseles ver que la lectura de artículos técnicos es necesaria para su autoformación, y que la mayor parte de éstos aparecen en publicaciones internacionales.

La lista de lecturas complementarias recomendada es:

"*Anecdotes/stories about Software Engineering*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/anecdote. [Última vez visitado, 28-1-2000]. July, 1993.

"*Computer Horror Stories*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/horror. [Última vez visitado, 28-1-2000]. July, 1993.

"*Origin of Term 'Software Engineering'*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/SEorigin. [Última vez visitado, 28-1-2000]. July, 1993.

"*Software Engineering Questions and Answers*". http://www.qucis.queensu.ca/FAQs/comp.software-eng/questions.html. [Última vez visitado, 28-1-2000]. 1998.

Amako, Katsuya. "*Object Modeling Technique - Summary Note*". http://arkhp1.kek.jp/managers/computing/activities/OO_CollectInfor/Methodologies/OMT/OMTBook/OMTBook.html. [Última vez visitado, 9-2-2000]. June, 1995.

Baber, Robert L. "*Comparison of Electrical 'Engineering' of Heaviside's Times and Software 'Engineering' of our Times*". IEEE Annals of the History of Computing, 19(4):5-17. 1997.

Bell, Alex E. and Schmidt, Ryan W. "*UMLoquent Expression of AWACS Software Design*". Communications of the ACM, 42(6):55-61. October, 1999.

Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

Boehm, Barry and Port, Dan. "*When Models Collide: Lessons from Software Systems Analysis*". IEEE IT Professional, 1(1):49-56. January/February, 1999.

Boehm, Barry W., Egyed, Alexander, Kwan, Julie, Port, Dan, Shah, Archita and Madachy, Ray. "*Using the WinWin Spiral Model: A Case Study*". IEEE Computer, 31(7):33-44, July, 1998.

Booch. Graddy. "*Quality Software and the UML*". Object Magazine. March, 1997.

Booch, Grady. "*Objectifying Information Technology*". Rational Software Corporation. http://www.rational.com [Última vez visitado, 1-12-1998]. 1998.

Brereton, Pearl, Budgen, David, Bennet, Keith, Munro, Malcom, Layzell, Paul, Macaulay, Linda, Griffiths, David and Stannett, Charles. "*The Future of Software*". Communications of the ACM, 42(12):78-84. December, 1999.

Brinkkemper, Sjaak, Hong, Shuguang, Bulthuis, Arjan and van den Goor, Geert. "*Object-Oriented Analysis and Design Methods a Comparative Review*". http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html. [Última vez visitado, 9-2-2000]. January, 1995.

Conallen, Jim. "*Modeling Web Application Architectures with UML*". Communications of the ACM, 42(10):63-70. October, 1999.

Chance, Brian D. and Melhart, Bonnie E. "*How to Develop Better System Requirements*". IEEE IT Professional, 1(3):70-72. May/June, 1999.

Chandra, Jagdish, March, Salvatore T., Mukherjee, Satyen, Pape, Will, Ramesh, R., Rao, H. Raghav and Waddoups, Ray O. "*Information Systems Frontiers*". Communications of the ACM, 43(1):71-79. January, 2000.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 29 de noviembre de 1999.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 18 de octubre de 1999.

Fayad, Mohamed E., Laitinen, Mauri and Ward, Robert P. "*Software Engineering in the Small*". Communications of the ACM, 43(3):115-118. March, 2000.

Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December, 1993.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Diagramas de Clase en UML 1.1*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(3):71-76. Marzo, 1998.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*Introducción al Análisis y Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(2):64-70. Febrero, 1998.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

Gardarin, Georges. "*Dominar las Bases de Datos*". Ediciones Gestión 2000, 1993.

Glass, Robert L. "*The Software Criris… Not?*". IEEE Computer, 27(4):104. April, 1994.

Gutiérrez, Inmaculada y Medinilla, Nelson. "*Contra el Arraigo de la Cascada*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. Pere Botella, Juan Hernández y Félix Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999.

Hawryszkiewycz, I. T. "*Introducción al Análisis y Diseño de Sistemas con Ejemplos Prácticos*". Anaya Multimedia, 1990.

Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

Hofmann, Hubert F., Pfeifer, Rolf and Vinkhuyzen, Erik. "*Situated Software Design*". Technical Report. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. 1993.

ICON Computing, Inc. "*A Comparison of OOA & OOD Methods*". Icon Computing Inc. http://www.iconcomp.com/papers/comp/. [Última vez visitado, 9-2-2000]. 1995.

Iivari, Juhani. "*Why Are CASE Tools Not Used*". Communications of the ACM, 39(10):94-103. October, 1996.

Jacobson, Ivar. "*Building Without Blueprints*". Object Magazine, 7(9): 71-72. November, 1997.

Kaindl, Hermann. "*Difficulties in the Transition from OO Analysis to Design*". IEEE Software, 16(5):94-102. September/October, 1999.

Kobryn, Cris. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October, 1999.

Kruchten, Philippe. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November, 1995.

Leaney, John. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. http://www.ee.uts.edu.au/~jrleaney/setut.htm. [Última vez visitado, 28-1-2000]. Octubre 1995.

Lewis, Ted. "*The Dark Side of Objects*". IEEE Computer, 27(12):6-7. December, 1994.

Lions, J. L. "*ARIANE 5 Flight 501 Failure*". Report by the Inquiry Board. http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html. [Última vez visitado, 28-1-2000]. París, 19 Julio 1996.

Miller, George A. "*The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*". The Psychological Review, Vol. 63: 81-97, 1956. Also available at http://www.well.com/user/smalin/miller.html [Última vez visitado, 3/8/1999].

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Ohnjec, Viktor. "*Converging on OOAD Agreement*". Applications Development Trends, 4(2). February, 1997.

Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

Perrochon, Louis and Mann, Walter. "*Inferred Designs*". IEEE Software, 16(5):46-51. September/October, 1999.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

Raccoon, L. B. S. "*Fifty Years of Progress in Software Engineering*". Software Engineering Notes (SEN), 22(1):88-104. January, 1997.

Rine, David. "*Object-Oriented Technology and Software Reuse*". IEEE Computer, 26(7):6. July, 1993.

Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

Rumbaugh, James. "*The OMT Process*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. May, 1994.

Rumbaugh, James. "*OMT: The Object Model*". Journal of Object-Oriented Programming, 7(8):21-27. January, 1995.

Rumbaugh, James. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming, 7(9):6-12. February, 1995.

Rumbaugh, James. "*What Is a Method?*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. October, 1995.

Rumbaugh, James. "*Bridging the Gap. Building Complex Systems by Leveling and Layering*". Rose Architect Magazine, 2(2). Winter, 1999.

Sharma, Srinarayan and Rai, Arun. "*CASE Deployment in IS Organizations*". Communications of the ACM, 43(1):80-88. January, 2000.

Silva, Manuel. "*Las Redes de Petri: En la Automática y la Informática*". Editorial AC, libros científicos y técnicos. Madrid. 1985.

Singh, Raghu. "*The Software Life Cycle Processes Standard*". IEEE Computer, 28(11):89-90. November, 1995.

Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

Whitlock, David. "*The Unified Modeling Language*". http://watson2.cs.binghamton.edu/~dwhitloc/uml/paper/part1.html. [Última vez visitado, 2-2-1999]. 1999.

Yourdon, Edward. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

Bibliografía para la preparación de las clases teóricas

Aunque se cuenta con un material ya desarrollado para impartir las clases teóricas de Ingeniería del Software, compuesto por unos apuntes y sus correspondientes transparencias [García, 1999], se dispone de una importante fuente de referencias bibliográficas de consulta, que permite que, tanto los apuntes como las clases, sean elementos vivos que evolucionan todos los cursos, realimentándose de las experiencias adquiridas por el profesor tanto en su labor docente como investigadora, así como por la lectura de estas referencias bibliográficas.

La Ingeniería del Software, como todo en el mundo de la Informática, evoluciona muy deprisa, lo que requiere que el profesor haga un esfuerzo extra para estar al tanto de dichos avances.

Por otra parte, la Ingeniería del Software tiene una vertiente práctica importante que justifica la búsqueda de *nuevos ejemplos, casos prácticos*... para aplicar en las clases teóricas y prácticas.

Así, la lista de referencias bibliográficas utilizadas para la preparación de las clases del programa de teoría de la asignatura de Ingeniería del Software, a mayores de las lecturas complementarias, es:

Abernethy, Ken and Kelly, John C. "*Comparing Object-Oriented and Data Flow Models – A Case Study*". In Proceedings of the 1992 ACM Computer Science 20th annual conference on Communications, CSC '92. (March 3-5, 1992, Kansas City, MO - USA). Pages 541-547. ACM. 1992.

Aldis, Margaret. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

Amescua Seco, Antonio de, García Sánchez, Luis, Martínez Fernández, Paloma y Díaz Pérez, Paloma. "*Ingeniería del Software de Gestión. Análisis y Diseño de Aplicaciones*". Paraninfo, 1995.

Appleton, Bradford D. "*A Software Design Specification Template*". http://www.enteract.com/~bradapp/docs/sdd.html. 1997.

Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

Battini, C., Ceri, S. and Navathe, S. B. "*Conceptual Database Design: An Entity Relationship Approach*". Benjamin/Cummings, 1992.

Bell, Doug, Morrey, Ian and Pugh, John. "*Software Engineering. A Programming Approach*". 2nd Edition. Prentice Hall, 1992.

Berard, Edward V. "*Be Careful with 'Use Cases'*". The Object Agency, Inc., 1995.

Blaha, Michael and Premerlani, William. "*Object-Oriented Modeling and Design for Database Applications*". Prentice-Hall, 1998.

Blaha, Michael and Premerlani, William. "*Object-Oriented Design of Database Applications*". Rose Architect, 1(2). January, 1999.

Blaha, Michael and Premerlani, William. "*Implementing UML Models with Relational Databases*". Rose Architect, 1(3). April, 1999.

Boehm, Barry W. "*Software Engineering Economics*". Prentice Hall, 1981.

Boehm, Barry W. and Bose, Prasanta. "*A Collaborative Spiral Software Process Model Based on Theory W*". In Proceedings of the Int'l Conf. Software Process. IEEE Computer Society Press. Pages, 59-68. 1994.

Booch, Grady. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996.

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. "*El Lenguaje Unificado de Modelado*". Object Technology Series. Addison Wesley, 1999.

Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1990.

Bruegge, Bernd and Dutoit, Allen H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Budgen, David. "*Introduction to Software Design*". SEI Curriculum Module SEI-CM-2-2.1. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1989.

Burns, Thomas, Fong Elizabeth, Jefferson, David, Knox, Richard, Mark, Leo, Reedy, Christopher, Reich, Louis, Roussopoulos, Nick and Truszkowski, Walter. "*Reference Model for DBMS Standardization*". SIGMOD RECORD, 15(1). March, 1986.

Cantor, Murray R. "*Object-Oriented Project Management with UML*". Wiley & Sons, 1998.

Cockburn, Alistair. "*Writing Effective Use Cases*". To be published by Addison Wesley Longman in 2000. Draft 2 available at http://members.aol.com/humansandt/crystal/usecasetechnique/getWEUCbook.htm. [Última vez visitado, 14-3-2000]. December, 1999.

Collins-Cope, Mark. "*The Requirements/Service/Interface (RSI) Approach to Use Case Analysis (A Pattern for Structured Use Case Development)*". Ratio Group Ltd. http://www.ratio.co.uk/RSI.htm. [Última vez visitado, 19-5-1999]. 1999.

Conger, Sue. "*The New Software Engineering*". Course Technology, 1994.

Champeaux, Dennis de (moderator), Constantine, Larry, Jacobson, Ivar, Mellor, Stephen, Ward, Paul and Yourdon, Edward. "*PANEL: Structured Analysis and Object Oriented Analysis*". In Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA90/ECOOP90. (October 21 - 25, 1990, Ottawa Canada). Pages 135-139. ACM, 1990.

Champeaux, Dennis de, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison-Wesley, 1993. HTML Edition available at http://gee.cd.oswego.edu/dl/oosdw3. [Última vez visitado, 1-2-2000].

Christel, Michael G. and Kang, Kyo C. "*Issues in Requirements Elicitation*". Technical Report CMU/SEI-92-TR-12 (ESC-TR-92-012). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). September, 1992.

Christerson, Magnus. "*From Use Cases to Components*". Rose Architect, 1(1). October, 1998.

Davis, Alan M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

Davis, William S. "*Herramientas CASE. Metodología Estructurada para el Desarrollo de los Sistemas*". Paraninfo, 1992.

Dedene, G. and Snoeck. "*MERODE: A Model-Driven Entity-Relationship Object-Oriented Development Method*". ACM Software Engineering Notes, 19(3):51-61. July, 1994.

Díaz, Oscar and Rodríguez, Juan José. "*Change Case Analysis*". Journal of Object-Oriented Programming (JOOP), 12(9):9-15,48. February, 2000.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel and Ruiz Cortés. "*An Object-Oriented Model and a CASE Tool for Software Requirements Management and Documentation*". In Proceedings of the 4th Workshop MENHIR. Francisco José García and José Manuel Marqués editors. (May 6-7, 1999, Sedano, Burgos – Spain). Pages 6-10. 1999.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel y Ruiz Cortés. "*Una Propuesta Metodológica para la Recolección de Requisitos de un Sistema Software*". En las Actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. (Murcia, 13 y 14 de Noviembre 1998). Páginas 37-48. 1998.

Durán Toro, Amador, Bernárdez Jiménez, Beatriz, Toro Bonilla, Miguel and Ruiz Cortés. "*Elicitación de Requisitos de Usuario Mediante Plantillas y Patrones de Requisitos*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. Pere Botella, Juan Hernández y Félix Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 183-194. 1999.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 29 de noviembre de 1999.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 18 de octubre de 1999.

D'Souza, Desmond F. and Wills, Alan Cameron. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

Ellison, Karen S. "*Developing Real-Time Embedded Software in a Market-Driven Company*". John Wiley & Sons, 1994.

Eriksson, Hans-Erik and Penker, Magnus. "*UML Toolkit*". John Wiley & Sons, 1998.

Fairley, Richard. "*Ingeniería del Software*". McGraw Hill, 1988.

Firesmith, Donald, Henderson-Sellers, Brian, Graham, Ian and Page-Jones, Meilir. "*OPEN Modeling Language (OML) Reference Manual*". Version 1.0. OPEN Consortium. December, 1996.

Firesmith, Donald G. and Henderson-Sellers, Brian. "*Upgrading OML to Version 1.1: Part 1 Referencial Relationships*". Journal of Object-Oriented Programming (JOOP), 11(3):48-57. June, 1998.

Firesmith, Donald G. and Henderson-Sellers, Brian. "*Upgrading OML to Version 1.1: Part 2 Additional Concepts and Notation*". Journal of Object-Oriented Programming (JOOP), 11(5):61-67. September, 1998.

Firesmith, D. and Henderson-Sellers, B. "*Improvements to the OPEN Process Metamodel*". Journal of Object-Oriented Programming (JOOP), 12(7):30-35. November/December, 1999.

Fisher, Alan S. "*Tecnología CASE. Herramientas de Desarrollo de Software*". 2ª Edición. Anaya Multimedia, 1993.

Fowler, Martin. "*A Survey of Object-Oriented Analysis and Design Techniques*". Technical-Report, 1997.

Fowler, Martin. "*Use and Abuse Cases*". Distributed Computing. April, 1998.

Fowler, Martin and Scott, Kendall. "*UML Gota a Gota*". Addison Wesley Longman (Pearson), 1999.

Frakes, William B., Fox, Christopher J. and Nejmeh, Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

Gaitero Gordillo, Domingo. "*Metodología Métrica. Un Enfoque Práctico*". Everest Multimedia, 1997.

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Ghezzi, Carlo, Jazayeri, Mehdi and Mandrioli, Dino. "*Fundamentals of Software Engineering*". Prentice-Hall International, 1991.

Graham, Ian. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Graham, Ian, Henderson-Sellers, Brian and Younessi, Houman. "*The OPEN Process Specification*". ACM Press Books. The OPEN Series. Addison-Wesley, 1997.

Gray, Lewis. "*ISO/IEC 12207 Software Lifecycle Processes*". Crosstalk. The Journal of Defense Software Engineering, 9(8). August, 1996.

Guttag, John. "*Abstract Data Types and the Development of Data Structures*". Communications of the ACM, 20(6):396-404. June, 1977.

Hatley, Derek J. and Pirbhai, Imtiaz. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

Henderson-Sellers, Brian. "*The OPEN-Mentor Methodology*". Object Magazine, 6(9):56-59. November, 1996. Available online at http://www.open.org.au/Publications/Documents/open.pdf [Última vez visitado, 20-3-2000].

Henderson-Sellers, Brian. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Henderson-Sellers, Brian. "*Choosing between UML and OPEN*". 1997.

Henderson-Sellers, Brian. "*OML: Proposals to Enhance UML*". In Proceedings of *<<UML>>'98 Beyond the Notation* Conference. (3rd-4th June 98, Mulhouse - France). June, 1998.

Henderson-Sellers, Brian and Graham, Ian. "*Process and Product Life Cycles: OPEN's Version 2 Life Cycle Model*". Journal of Object-Oriented Programming (JOOP), 13(1):23-26,39. March/April, 2000.

Henderson-Sellers, Brian, **Simons, Tony and Younessi, Houman**. "*The OPEN Toolbox of Techniques*". Open Series. Addison-Wesley, 1998.

Hix, D. and Hartson, H. R. "*Developing User Interfaces: Insuring Usability through Product and Process*". John Wiley & Sons, 1993.

Hofmann, Hubert F. "*Requirements Engineering*". Technical Report 93.05. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. March, 1993.

Iturrioz Sánchez, Juan Ignacio. "*Una Metodología para el Desarrollo de Sistemas de Base de Datos Objeto-Relacional*". Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos. Universidad del País Vasco – Euskal Herriko Unibertsitatea. Junio, 1998.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Jacobson, Ivar, Griss, Martin and Jonsson, Patrik. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley, 1997.

Jones, Trevor H., Song, Il-Yeol and Park, E. K. "*Ternary Relationship Decomposition and Higher Normal Form Structures Derived from Entity Relationship Conceptual Modeling*". In Proceedings on 1996 ACM on Computer science conference, CSC '96. (Feb. 16-18, 1996, Philadelphia, PA - USA). Pages 96-104. ACM. 1996.

Kaman Sciences Corporation. "*A State of the Art Report: Software Design Methods*". Kaman Sciences Corporation. March, 1994.

Karma, Gerald M. and Casselman, Ronald S. "*A Cataloging Framework for Software Development Methods*". IEEE Computer, 26(2):34-45. February, 1993.

Kenworthy, Edward. "*Use Case Modelling. Capturing User Requirements*". http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.html. [Última vez visitado, 2-2-1999]. December, 1997.

Kowal, James A. "*Behavior Models. Specifying User's Expectations*". Prentice-Hall International, 1992.

Kruchten, Philippe. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November, 1995.

Kruchten, Philippe. "*A Rational Development Process*". Crosstalk, 9(7):11-16. July, 1996.

Kruchten, Philippe. "*A Rational Development Process*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 4-2-1999]. 1996.

Kruchten, Philippe. "*The Rational Unified Process. An Introduction*". Object Technology. Series Addison-Wesley, 1999.

Larman, Craig. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Liberty, Jesse. "*Beginning Object-Oriented Analysis and Design with C++*". Wrox Press Ltd., 1998.

Long, Fred and Morris, Ed. "*An Overview of PCTE: A Basis for a Portable Common Tool Environment*". Technical Report CMU/SEI-93-TR-1, ESC-TR-93-175. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., March, 1993.

López, Natalie, Migueis, Jorge y Pichon, Emmanuel. "*Integrar UML en los Proyectos*". Gestión 2000, 1998.

Lucero, José Luis. "*Gestión de la Calidad del Software*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Abril 1996.

Lucero, José Luis y Ramos, Miguel Ángel. "*Dirección de Departamentos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Febrero-Marzo 1996.

Lucero, José Luis y Ramos, Miguel Ángel. "*Gestión de Proyectos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Enero-Febrero 1996.

Lunn, Ken. "*Object Oriented Analysis and Design – Course Notes*". 1997.

Marqués Corral, José Manuel. "*Diseño de Interfaces de Usuario*". Apuntes de la asignatura Ingeniería del Software II de la Ingeniería Técnica de Informática de Sistemas. Universidad de Valladolid.

Martin, James and Odell, James J. "*Métodos Orientados a Objetos: Conceptos Fundamentales*". Prentice Hall, 1997.

Martin, James and Odell, James J. "*Métodos Orientados a Objetos: Consideraciones Prácticas*". Prentice Hall Hispanoamericana, 1997.

Matheron, Jean-Patrick. "*Merise - Metodología de Desarrollo de Sistemas. Casos Prácticos*". Paraninfo, 1990.

Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999.

Miguel, Adoración de y Piattini, Mario G. "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al Modelo Relacional*". Ra-ma, 1993.

Miguel, Adoración de y Piattini, Mario. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Moore, Jim. "*ISO 12207 and Related Software Life-Cycle Standards*". http://www.acm.org/tcs/lifecycle.html. [Última vez visitado, 12-3-2000]. 1996.

Muller, Pierre-Alain. "*Modelado de Objetos con UML*". Ediciones Gestión 2000, 1997.

Nerson, J. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September, 1992.

Newman, M. and Lamming, G. "*Interactive System Design*". Addison-Wesley, 1995.

Odell, James J. "*Advanced Object-Oriented Analysis and Design Using UML*". Cambridge University Press. SIGS Books, 1998.

Oestereich, Bernd. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Parnas, David L. "*Designing Software for Ease of Extension and Contraction*". IEEE Transactions on Software Engineering, SE-5(2):128-138. March, 1979.

Pfleeger, Shari Lawrence. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Piattini Velthuis, Mario Gerardo. "*Ciclos de vida para Sistemas Orientados a Objetos*". Cuore, (7):6-11. Septiembre, 1995.

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Piattini, Mario G., Calvo-Manzano, José A., Cervera Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Piattini Velthuis, Mario G. y Daryanani Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

Pooley, Rob and Stevens, Perdita. "*Using UML Software Engineering with Objects and Components*". Addison-Wesley, 1999.

Premerlani, William J., Blaha, Michael R., Rumbaugh, James E. and Varwing, Thomas A. "*An Object-Oriented Relational Database*". Communications of the ACM, 33(11):99-109. November, 1990.

Pressman, Roger S. "*Ingeniería del Software: Un Enfoque Práctico*". 3ª Edición, McGraw Hill, 1993.

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición, McGraw Hill, 1998.

Raghavan, S., Zelesnik, G. and Ford, G. "*Lecture Notes on Requirements Elicitation*". Educational Materials. CMU/SEI-94-EM-10. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March, 1994.

Rational Software Corporation. "*Unified Modeling Language for Real-Time Systems Design*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 16/6/97]. 1997.

Rational Software Corporation. "*A Rational Approach to Software Development Using Rational Rose 4.0*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 1/12/98]. 1998.

Rational Software Corporation. "*Rational Unified Process. Best Practices for Software Development Teams*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 1/12/98]. 1998.

Rational Software Corporation. "*Reaching CMM Levels 2 and 3 with the Rational Unified Process*". Rational Software Corporation. http://www.rational.com. [Última vez visitado, 12-2-2000]. 1998.

Rational Software Corporation. "*Inside the Unified Modeling Language - UML*". Multimedia Educational Tool. April, 1999.

Rational Software Corporation. "*Rational Unified Process. Product Information FAQ*". Rational Software Corporation. http://www.rational.com. [Última vez visitado, 8-2-2000]. 2000.

Rational Software Corporation and Context Integration. "*Building Web Solution with the Rational Unified Process: Unifying the Creative Design Process and the Software Engineering Process*". White Paper. http://www.rational.com. 1999.

Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

Rivas, Erick, DeSilva, Dilhar, McDaniel, Terrie and Atkinson, Colin. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January, 1997.

Rivero Cornelio, E. "*Bases de Datos Relacionales*". Paraninfo, 1991.

Robinson, Peter J. "*Hierarchical Object-Oriented Design*". Object-Oriented Series. Prentice-Hall, 1992.

Rosenberg, Doug and Scott, Kendall. "*Use Case Driven Object Modeling with UML. A Practical Approach*". Object Technology Series. Addison-Wesley, 1999.

Rumbaugh, James. "*Disinherited! Examples of Misuse of Inheritance*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. February, 1993.

Rumbaugh, James. "*Getting Started. Using Use Cases To Capture Requirements*". Journal of Object-Oriented Programming (JOOP), 7(5):8-12, 23. September, 1994.

Rumbaugh, James. "*Building Boxes: Subsystems*". Journal of Object-Oriented Programming, 7(6): 16-21. October, 1994.

Rumbaugh, James. "*Driving to a Solution. Reification and the Art of System Design*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. July, 1995.

Rumbaugh, James. "*A Private Workspace. Why a Shared Repository Is Bad for Large Projects*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. September, 1995.

Rumbaugh, James. "*Taking Things in Context. Using Composites to Build Models*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. November, 1995.

Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

Rumbaugh, James. "*The View from the Front: Changes to UML by the Revision Task Force*". Rose Architect, 1(3). Summer, 1999.

Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

Sawyer, Pete and Kotonya, Gerald. "*SWEBOK: Software Requirements Engineering Knowledge Area Description*". In [Abran et al., 1999], 1999.

Scacchi, Wait. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). October, 1987.

Schneider, Geri and Winters, Jason P. "*Applying Use Cases. A Practical Guide*". Object Technology Series. Addison-Wesley, 1998.

SEI Requirements Engineering Project. "*Requirements Engineering and Analysis. Workshop Proceedings*". Technical Report CMU/SEI-91-TR-30 (ESD-TR-91-30). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). December, 1991.

Selic, Bran. "*Turning Clockwise: Using UML in the Real-Time Domain*". Communications of the ACM, 42(10):46-54. October, 1999.

Selic, Bran and Rumbaugh, James. "*Using UML for Modeling Complex Real-Time Systems*". Technical Report. March, 1998.

Senn, James A. "*Análisis y Diseño de Sistemas de Información*". 2ª Edición. McGraw-Hill, 1992.

Shneiderman, B. "*Designing the User Interface. Strategies for Effective Human-Computer Interaction*". 2nd Edition. Addison-Wesley, 1992.

Sibley, Edgar H. "*Entity-Life Modeling and Structured Analysis in Real-Time Software Design – A Comparison*". Communications of the ACM, 32(12):1458-1466. December, 1989.

Sommerville, Ian. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

Telefónica. "*MARTE: Metodología Armonizada de Telefónica*". Telefónica. 1998.

Texel, Putnam P. and Williams, Charles B. "*Use Cases Combined with Booch, OMT UML. Process and Products*". Prentice Hall, 1997.

Tkach, Daniel, Fang, Walter and So, Andrew. "*Visual Modeling Technique*". Addison-Wesley, 1996.

TOA. "*A Comparison of Object-Oriented Methodologies*". The Object Agency, Inc. 1995.

Tomayko, James E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

Vadaparty, Kumar. "*Use Cases - Basics*". Journal of Object-Oriented Programming (JOOP), 12(9). February, 2000.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction & Tools*". Prentice-Hall, 1985.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 2: Essential Modeling Techniques*". Prentice-Hall, 1985.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 3:Implementation Modeling Techniques*". Yourdon Press, 1986.

Warmer, Jos and Kleppe, Anneke. "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley. Object Technoloy Series, 1999.

Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice Hall, 1990.

Yourdon, Edward. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

Yourdon, Edward and Argila, Carl. "*Case Studies in Object Oriented Analysis & Design*". Yourdon Press Computing Series. Prentice Hall, 1996.

Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.

Yourdon, Edward, Whitehead, Katharine, Toman, Jim, Oppel, Karin and Nevermann, Peter. "*Mainstream Objects. An Analysis and Design Approach for Business*". Yourdon Press, 1995.

Zhang, David D. "*Use Case Modeling for Real-time Application*". In Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'99. (27 - 29 January, 1999, Santa Barbara, California – USA). Pages 56-64. IEEE Computer Society, 1999.

Zhao, Liping and Foster, Ted. "*Modeling Roles with Cascade*". IEEE Software, 16(5):86-93. September/October, 1999.

5.2.2 Programa de la parte práctica

La parte práctica de la asignatura de Ingeniería del Software está orientada para satisfacer aquellos objetivos prácticos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**P1, P2** y **P4**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000] (*incluido en el Apéndice A de este Proyecto Docente*).

P1	Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.
P2	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
P4	Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Tabla 5.17. Objetivos del programa de prácticas de la asignatura Ingeniería del Software

5.2.2.1 Consideraciones iniciales

El objetivo fundamental de las prácticas de la asignatura de Ingeniería del Software es el modelado de sistemas software. Además, se considera que para aprender a modelar es más efectivo la realización de talleres donde se realice un ejercicio y se discuta su solución, en lugar de hacer más hincapié en el manejo de las herramientas CASE; manejo que por otra parte pueden aprender y entrenarse en horas de práctica libre.

Se considera que las 15 horas, que equivalen al crédito y medio de las prácticas, deben repartirse en prácticas presenciales, talleres fundamentalmente, y prácticas libres, que deben aprovechar para hacer los informes de los talleres y la práctica obligatoria.

Para el mejor aprovechamiento de las clases presenciales, éstas se llevan a cabo cada 15 días durante dos horas consecutivas.

Una contrariedad es tener que contar con los conocimientos teóricos necesarios para realizar las prácticas. Esto obliga a que el primer taller debe basarse en conocimientos previos, *modelo entidad-relación*, adquiridos en la asignatura de **Diseño de Bases de Datos**, impartida en el segundo curso. También por restricciones temporales, sólo se podrá realizar un taller dedicado a la tecnología de objetos.

5.2.2.2 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte práctica de esta asignatura se ha organizado en cinco prácticas, repartidas en tres grupos, como se muestra en la Tabla 5.18.

Talleres (10 Horas)
Práctica 1. Taller de modelado de datos. Repaso al modelo entidad-relación (2H)
Práctica 2. Taller de modelado funcional de sistemas (método clásico) (2H)
Práctica 3. Taller de modelado funcional de sistemas (método de Yourdon) (4H)
Práctica 4. Taller de Orientación al Objeto (2H)
Laboratorio (2 Horas)
Práctica 5. Manejo de una herramienta CASE (2H)
Práctica obligatoria (Prácticas libres)
Práctica 6. Realización de una ERS y un prototipo de una aplicación software

Tabla 5.18. Estructura del programa de prácticas de la asignatura Ingeniería del Software (1,5 créditos)

En la Tabla 5.19 se presenta la correspondencia existente entre el programa de prácticas y los objetivos prácticos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Práctica 1	P1, H1, H2, H4
Práctica 2	P1, H1, H2, H4
Práctica 3	P1, H1, H2, H4
Práctica 4	P2, H1, H2, H4
Práctica 5	P4
Práctica 6	P1, P2, P4, H1, H2, H4

Tabla 5.19. Correspondencia entre el temario de prácticas y los objetivos prácticos de la asignatura

Desarrollo de las clases prácticas (talleres)

Los talleres se llevan a cabo en la clase de teoría. Los alumnos se agrupan en grupos de entre cuatro y seis personas, que durante, aproximadamente una hora, de las dos de las que disponen, el grupo discute y desarrolla la solución a un problema cuyo enunciado, previamente, ha sido facilitado a través de la página de la asignatura.

Uno de los grupos se ofrecerá voluntario para presentar la práctica, recibiendo como recompensa 0,75 puntos, sumados a la nota final.

Durante la segunda hora, el grupo desarrolla su solución en la pizarra (si la solución del problema tuviera una gran cantidad de diagramas, se dedican las dos horas para su solución, y se aplaza la discusión para la siguiente clase, utilizándose transparencias).

Tras su exposición, se desarrolla un debate con el resto de los grupos, moderado por el profesor. De esta manera se logra, por una parte, que los alumnos se expresen en público y defiendan su trabajo, y por otra la corrección pública de los errores cometidos en los modelos.

Con los comentarios, el grupo encargado de la defensa elabora un informe que entregan al profesor, quien, tras comprobar que no contiene errores, lo publicará en la página de la asignatura para que sirva de material de estudio al resto de los alumnos, y para promociones futuras.

Evaluación de la parte práctica

La forma principal de evaluar la parte práctica de esta asignatura es mediante la realización de una práctica obligatoria, cuyos requisitos deben obtenerse de *usuarios* o *clientes* reales.

En la Figura 5.5 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota de la práctica obligatoria y los informes de los talleres, realizados voluntariamente.

Si (Teoría ≥ 4,75) **y** (***Práctica*** ≥ 5.0)
    Nota Final = (Teoría*0,5) + (***Práctica****0,5) + ***Nota trabajos***
Sino
    ø
Fin si

Figura 5.5. Influencia de la nota en la parte práctica en la nota final de Ingeniería del Software

Bibliografía básica de referencia

De la lista de títulos recomendados, los siguientes pueden ser los más adecuados para ser consultados en la realización de la parte práctica.

- **Booch, Grady, Rumbaugh, James and Jacobson, Ivar.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **OMG.** "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.
- **Rumbaugh, J., Jacobson, I. and Booch, G.** "*The Unified Modeling Language Reference Manual*". Addison-Wesley. 1999.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

A los que habría que añadir alguno donde se aborde la creación de modelos entidad-interrelación, como por ejemplo:

- **Miguel, Adoración de y Piattini, Mario G.** "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al Modelo Relacional*". Ra-ma, 1993.
- **Miguel, Adoración de y Piattini, Mario**. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

5.2.2.3 Desarrollo comentado del programa

En el programa de práctica se distinguen tres bloques. El primero está compuesto por los talleres, donde se va a hacer hincapié en los aspectos de modelado. El segundo sería el laboratorio, donde se puede aprovechar para introducir alguna herramienta CASE, ya sea orientada al paradigma estructurado, como **Easy Case**, u orientada al paradigma

objetual, como **Rational Rose**. El tercero, y último, es el que se ocupa de la práctica obligatoria, por lo tanto práctica no presencial.

Talleres

Los talleres constituyen la parte práctica de la asignatura donde la relación docente-discentes es más importante.

Los objetivos generales de estos talleres se pueden resumir en los siguientes puntos:

- Practicar el modelado de sistemas software tanto en el paradigma estructurado como en el objetual.
- Comentar en público los errores más frecuentes que se cometen a la hora de realizar los modelos.
- Potenciar la comunicación en público de los alumnos.
- Obligar a la realización de informes que documenten su labor.
- Incentivar la participación del alumno en el desarrollo de la asignatura.

Idealmente cada alumno matriculado en la asignatura participará en tres talleres de dos horas de duración cada uno, más en un cuarto de cuatro horas de duración; lo cual supone ocupar aproximadamente el **67%** del tiempo oficial de prácticas en estas actividades.

Los talleres a realizar son:

- ***Taller 1:*** Modelo de información – Entidad-Interrelación.
- ***Taller 2:*** Modelo funcional – Enfoque clásico.
- ***Taller 3:*** Modelo funcional – Enfoque de Yourdon.
- ***Taller 4:*** Diagramas de clases

A continuación se va a desarrollar de una forma más detallada cada uno de ellos.

Taller 1: Modelo de información

Es el primero que se realiza. Sirve como repaso a las técnicas de modelado conceptual de bases de datos relacionales impartidas en la asignatura de **Diseño de Bases de Datos** del segundo curso.

Permite al profesor realizar prácticas mientras se avanza en el temario teórico para poder abordar las prácticas con los DFDs.

El modelo sobre el que se trabaja es el entidad-interrelación, aunque además del modelo conceptual, se les pide que construyan el modelo lógico (*modelo relacional*) equivalente al modelo conceptual elaborado, de manera que éste se encuentre en forma normal de Boyce/Codd.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Taller 2: **Modelo funcional – Enfoque clásico**

En este segundo taller se pretende que los alumnos se familiaricen con la creación de DFDs nivelados, su balanceo y la identificación de las estructuras inválidas de esta técnica.

Este primer taller dedicado al modelado funcional, se realiza utilizando un enfoque clásico, totalmente descendente, para lo cual se requiere un problema relativamente pequeño.

Se justifica un taller de estas características por los siguientes motivos:

- El objetivo fundamental es que el alumno se familiarice con la técnica.
- El alumno está acostumbrado a trabajar con técnicas descendentes.
- Es una buena oportunidad para comentar los problemas del modelado descendente en la creación de DFDs.

El informe final a realizar deberá incluir además el diccionario de datos y la especificación de los procesos primitivos.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Taller 3: **Modelo funcional – Enfoque de Yourdon**

El tercer taller también se dedica al modelado funcional, pero ahora haciendo uso del método de Yourdon.

Se pretende que el grupo que defienda la práctica muestre la evolución desde el DFD preliminar hasta el DFD nivelado final. Como esto supone realizar muchos diagramas, y el nivelado ascendente/descendente de este método les resulta más complejo, debido a su dependencia de las técnicas descendentes, a este taller se le dedican cuatro horas, dos para la elaboración y dos para la presentación y la discusión.

El informe final a realizar deberá incluir además el diccionario de datos y la especificación de los procesos primitivos.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Taller 4: **Diagramas de clases**

El cuarto y último taller se dedica a la Orientación a Objetos. Por restricciones temporales, cuando se estudian las técnicas orientadas al objeto, UML, queda muy poco tiempo para finalizar la asignatura, y por tanto para llevar a cabo el taller. Así, hay que decantarse por una técnica, y quizás la más representativa sea el diagrama de clases de UML.

Lo más interesante de esta práctica es que, partiendo de un enunciado pequeño que represente un caso conocido por los alumnos, se vayan identificando las clases, relacionándolas y refinándolas, hasta un punto no demasiado elaborado, pero lo suficiente para que capten la filosofía de trabajo.

El informe final debe incluir la documentación de cada clase y cada relación, utilizando para ello las plantillas recomendadas en [Durán y Bernárdez, 1999a].

Este taller busca satisfacer el objetivo **P2**, *aplicar de forma práctica los conceptos teóricos de Orientación a Objetos*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Laboratorio

De cara a satisfacer el objetivo **P4** de la Unidad Docente de Ingeniería del Software y Orientación a Objetos, *utilización de herramientas CASE para la gestión y desarrollo de sistemas software*, se puede dedicar una jornada de prácticas, aproximadamente el 13% de la parte de prácticas, para introducir alguna herramienta CASE, ya sea dedicada al paradigma estructurado, cada vez más escasas y con licencias muy costosas, como **Easy CASE**, o dedicadas al paradigma objetual, concretamente a UML, muy abundantes en Internet, ya sea por ser de libre distribución como **ArgoUML**, o versiones reducidas, como puede ser el caso de **Rational Rose 98** o **Rational Rose 2000** (http://www.rational.com).

Práctica obligatoria

Es la base fundamental para la evaluación de la parte práctica de la asignatura de Ingeniería del Software.

En las primeras semanas de la asignatura se publica en la página de la asignatura en qué consiste y la fecha de entrega.

La práctica debe hacerse en grupos, compuestos entre tres y cinco alumnos, y ellos mismos deben buscarse un *cliente real* a quién hacerle las entrevistas oportunas para la obtención de los requisitos de la práctica.

Esta forma de plantear la práctica obligatoria tiene las siguientes ventajas:

- Se obliga a que los alumnos se acerquen a lo que es el mundo real [Nachbar, 1998]. Deben poner en práctica las técnicas de entrevista, y sufren lo difícil que es la comunicación con los clientes.
- Que cada grupo tenga un cliente distinto evita el plagio de prácticas.
- Potencia el trabajo en grupo. Se les da libertad para que ellos se organicen.
- Se les da libertad para elegir entre el paradigma estructurado o el objetual.

Como inconvenientes se pueden citar:

- Normalmente, el cliente es un familiar de alguno de los miembros del grupo, con lo que la entrevista puede ser realizada por una persona sólo, en lugar de participar todo el grupo.
- Debe ponerse un límite para que el grupo no minimice los requisitos a simples altas, bajas y modificaciones de unas entidades.
- Todos los inconvenientes que tiene el trabajo en grupo, cuando se hace un reparto del problema y no se trabaja en grupo, o cuando alguien decide *camuflarse* en el grupo y dejar que trabajen por él. Pero estos son problemas reales, que ellos mismos deben aprender a sortearlos.

Los productos que cada grupo debe entregar son dos:

1. *Un documento de* ***especificación de requisitos:*** El documento que recoja la especificación de requisitos puede basarse en la estructura de cualquiera de los formatos vistos en la parte teórica de la asignatura, debiendo incluir de forma obligatoria los siguientes aspectos:
   - *Una descripción textual describiendo los requisitos (catálogo de requisitos).*
   - *Modelo funcional.*
   - *Modelo de datos.*
   - *Modelo de comportamiento (si ha lugar).*
   - *Modelo lógico de datos (modelo relacional en FNBC).*
2. *Un* ***prototipo:*** Se debe realizar un prototipo de la aplicación que recoja toda la interfaz de usuario de la aplicación y un mínimo de la funcionalidad especificada. La funcionalidad implementada, así como su complejidad, influirá en la evaluación de la práctica. El prototipo se realizará utilizando el entorno de desarrollo visual que el grupo de trabajo estime oportuno, con la única restricción de que el prototipo se pueda ejecutar bajo **Windows NT** o **Linux**.

Cada grupo deberá entregar todos los ficheros de su trabajo organizados en documentación y ejecutables, así como la especificación de requisitos impresa.

Después de la entrega se publicará un calendario para la defensa por grupos de la práctica.

La defensa de la práctica se realizará en un examen oral, donde los miembros del grupo dispondrán de unos 5 minutos para presentar el prototipo, relacionándolo con el catálogo de requisitos establecido, su viabilidad... a semejanza de una reunión con un cliente que tiene que validar la especificación realizada. Durante aproximadamente otros 15 minutos los integrantes del grupo contestarán a preguntas sobre las características técnicas de los modelos realizadas por el responsable de la asignatura.

Al ser un trabajo realizado en grupo, todos los integrantes del mismo recibirán la misma nota. Esto significa que la actuación individual de cada integrante repercutirá en el global del grupo.

Con el fin de promover una mayor motivación hacia el trabajo, y por transitividad hacia la asignatura, la nota final del trabajo dependerá de un baremo impuesto en función de la calidad técnica y de la presentación de los trabajos. De todas formas, con independencia de la nota final, para que la práctica se considere superada se hará uso de la siguiente fórmula:

$$(NotaTécnica*0,8)+(Prototipo*0,1)+(PresentaciónOral*0,05)+(PresentaciónDocumentación*0,05) \geq 5$$

Para la realización de la práctica, se les da libertad sobre los métodos a seguir, siempre y cuando los resultados sean correctos.

Se hace hincapié en la utilización de estándares para realizar los documentos, aunque hasta ahora se les ha dado libertad para elegir los estándares a seguir, aunque en futuros cursos se irán desarrollando guías de estilo y contenido para la asignatura, semejantes a [Marqués, 1999] y [Tuya, 1999].

En general, con esta práctica se busca satisfacer en general todos los objetivos prácticos identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

De forma más concreta se podían citar los siguientes objetivos específicos:

- *Que los alumnos se enfrenten a la especificación (con uso de prototipos) de un caso real.*
- *Que los alumnos entiendan la diferencia entre un programa y un producto software.*
- *Que los alumnos pongan en práctica las técnicas explicadas en la parte teórica y en los talleres de la asignatura.*

- *Que los alumnos participen en un trabajo en grupo, donde hay roles diferenciados, y además debe existir una comunicación intensa entre todos ellos.*
- *Que los alumnos se acostumbren a documentar el software realizado, cogiendo soltura en la redacción de documentos técnicos.*
- *Que los alumnos manejen bibliografía especializada para profundizar en los aspectos teóricos que dan soporte a la práctica.*
- *Que los alumnos desarrollen y mejoren su capacidad de comunicación entre compañeros y ante un cliente real.*

5.3 Programa de la asignatura de Programación Orientada a Objetos

La asignatura de Programación Orientada a Objetos introduce las técnicas de programación orientadas al objeto en el currículo de los ingenieros técnicos en Informática de Sistemas en la Universidad de Salamanca.

Asignatura	*Programación orientada a objetos (optativa)*
Créditos	*3T + 3P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*2º*
Responsable	*Francisco José García Peñalvo (fgarcia@gugu.usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla 5.20. Ficha de la asignatura Programación Orientada a Objetos

Esta asignatura se imparte en el sexto cuatrimestre, es decir, en el segundo cuatrimestre del tercer curso, dentro del plan de estudios del B.O.E de 4-11-1997.

Esta asignatura está dotada de ***6 créditos, 3 teóricos*** y ***3 prácticos***. Se orienta como una continuación de la asignatura de *Ingeniería del Software*, centrándose en los aspectos de diseño e implementación orientados al objeto.

La primera vez que se imparte esta asignatura dentro del actual plan de estudios es en el curso académico 1999-2000, siendo el responsable de la misma el encargado de elaborar tanto el programa de teoría como el de prácticas.

Para la elaboración del programa se ha optado por una estrategia sustentada en los siguientes puntos:

1. La inclusión de la Orientación a Objetos y de la reutilización del software en el currículo de los ingenieros en informática se hace necesaria para afrontar la demanda y las características de los sistemas software actuales [Tewari, 1995].
2. El objetivo principal de esta asignatura es el diseño e implementación de software utilizando facilidades orientadas al objeto (*clases, plantillas, herencia, polimorfismo...*).

3. El énfasis se hace en el diseño, utilizando los lenguajes de programación orientados a objetos como el mecanismo para llevar a la práctica los diseños realizados, no como el fin de la asignatura.
4. Si se revisa la bibliografía, existe una gran diversidad en el lenguaje utilizado para la docencia de las técnicas de programación orientadas al objeto [DeClue, 1996]. En el programa propuesto prima el pragmatismo al utilizar C++ principalmente, y Java como complemento, ante otros enfoques más puristas, pero menos orientados a las necesidades industriales actuales, basados en Smalltalk [Bellin, 1992], [Luker, 1994] o Eiffel [Prieto, 1999]. Una interesante comparativa sobre los lenguajes más utilizados en la docencia de la Programación Orientada a Objetos es [Kölling, 1999a].
5. La visión que se da de los lenguajes es la de su modelo objeto; buscando la independencia de los diferentes entornos comerciales que lo soporten. En definitiva, se busca ofrecer una buena base teórica que evite caer en la trampa de las modas tecnológicas, que conducen a que un estudiante o profesional quede obsoleto junto a la moda tecnológica de turno [Denning, 1992]; es diferente enseñar los principios de programación orientada a objetos que enseñar un lenguaje de programación [Knudsen and Madsen, 1988]. En [Kölling, 1999b] se tiene una comparativa de diferentes entornos de desarrollo para lenguajes orientados a objetos.
6. Aunque la asignatura se fundamenta en la transmisión de conocimiento técnico, no se puede (ni se quiere) perder la oportunidad de hacer que los alumnos desarrollen y potencien otras habilidades más generales, pero de importancia capital en su futuro como profesionales: *acostumbrarse a consultar bibliografía (especialmente en inglés), haciendo hincapié en la importancia que tiene leer con cuidado para sintetizar, escribir y modelar de forma adecuada* [Jackson, 1995]; *escribir documentos técnicos que describan los diferentes elementos software que se crean a lo largo de un proyecto* [Deveaux et al., 1999] *cuidando los estándares de documentación* [Gersting, 1994], [McCauley et al., 1996], *sin que ello signifique dar la espalda a las reglas gramaticales y de estilo que ofrece un lenguaje tan rico como el castellano* [Vaquero, 1999]; *desarrollar una capacidad de comunicación oral adecuada* [McDonald and McDonald, 1993], [Fell et al., 1996].
7. Llegar a un equilibrio entre la teoría y la práctica [Glass, 1996], de forma que una base teórica bien establecida sea el fundamento adecuado para la aplicación práctica de la Ingeniería del Software.
8. Se parte de los siguientes prerrequisitos:

a. El alumno debe estar familiarizado con la teoría y la práctica del diseño y codificación en lenguajes procedurales, recomendablemente C [Kernighan and Ritchie, 1988]. Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos**.

b. Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

c. El alumno debe estar habituado a seguir un enfoque sistemático en el desarrollo de software, tal y como se ha explicado en la asignatura de *Ingeniería del Software*, además de conocer los principios básicos de modelado con UML, también impartidos en dicha asignatura.

5.3.1 Programa de la parte teórica

La parte teórica de la asignatura Programación Orientada a Objetos está orientada a satisfacer aquellos objetivos teóricos que, habiendo sido identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y al contexto docente de la asignatura (**T7** y **T9**), así como al objetivo práctico **P2**; además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000].

T7	Método de análisis/diseño orientado a objetos.
T9	Estudio y comprensión de los fundamentos del diseño de sistemas software.
P2	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

Tabla 5.21. Objetivos del programa de teoría de la asignatura Programación Orientada a Objetos

5.3.1.1 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de siete temas, organizados en tres unidades docentes, tal y como se puede apreciar en la Tabla 5.22.

La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve presentación del temario de teoría y práctica de la misma, haciendo énfasis en los objetivos, desarrollo y evaluación que se pretenden desarrollar.

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (4 Horas)
Tema 1. Lenguajes de programación orientados a objetos (2 Horas)
Tema 2. Orientación a Objetos y reutilización del software (2 Horas)
Unidad Docente II: Diseño orientado a objetos básico (14 Horas)
Tema 3. Técnicas básicas de diseño orientado a objetos (8 Horas)
Tema 4. Genericidad (4 Horas)
Tema 5. Manejo de excepciones (2 Horas)
Unidad Docente III: Diseño orientado a objetos avanzado (11 Horas)
Tema 6. Patrones de diseño (8 Horas)
Tema 7. Diseño por contrato (3 Horas)

Tabla 5.22. Estructura del programa de teoría de la asignatura Programación Orientada a Objetos

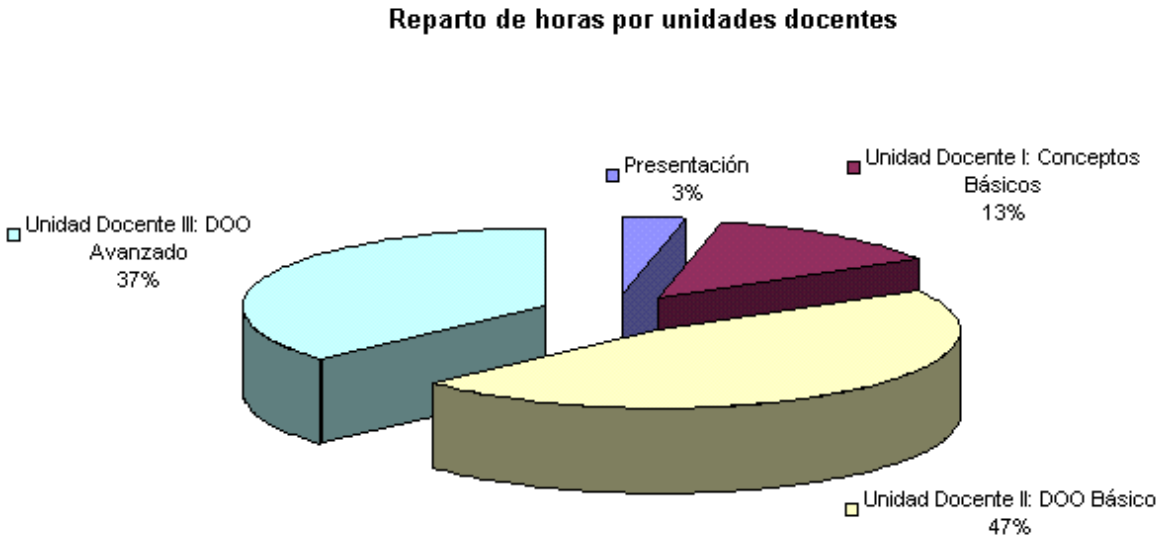

Figura 5.6. Reparto de las horas teóricas de Programación Orientada a Objetos entre las unidades docentes

En la Figura 5.6 y en la Figura 5.7 se puede apreciar como se reparten las horas del temario teórico de esta asignatura en sus unidades docentes y en sus temas.

En la Tabla 5.23 se presenta la correspondencia existente entre el programa de teoría y los objetivos teóricos perseguidos en esta asignatura.

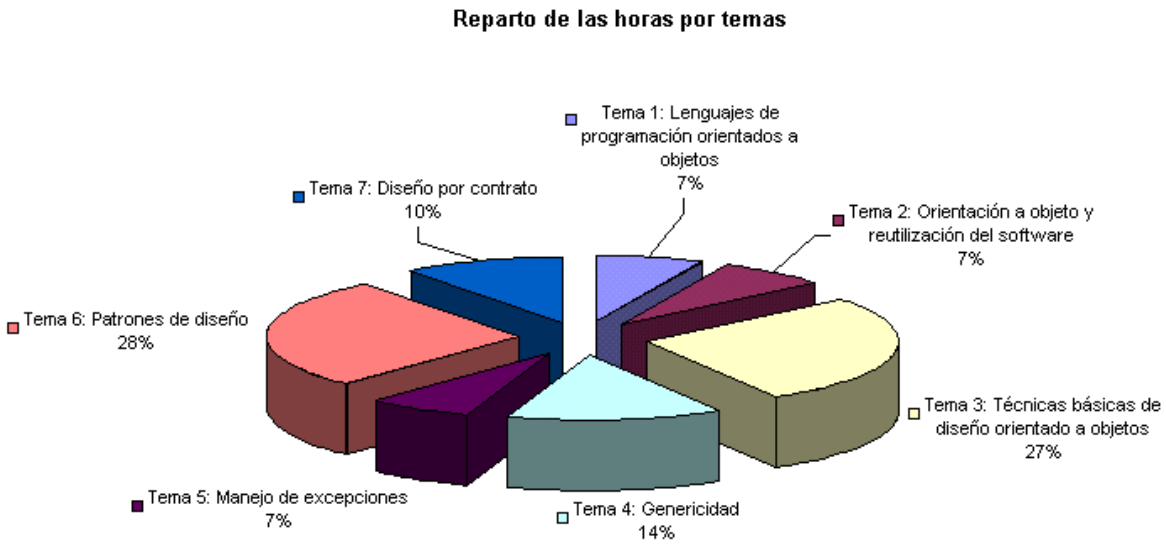

Figura 5.7. Reparto de las horas de teoría de Programación Orientada a Objetos entre los temas

Elemento Docente	Objetivos
Tema 1	T7
Tema 2	T7
Tema 3	T7, T9, P2
Tema 4	T7, T9, P2
Tema 5	T7, T9, P2
Tema 6	T7, T9, P2
Tema 7	T7, T9, P2

Tabla 5.23. Correspondencia entre el temario teórico y los objetivos teóricos de la asignatura

Estos temas pueden verse completados por otras actividades complementarias, que se llevarán a cabo dependiendo de diversos factores, entre los que cabe citar *la disponibilidad de tiempo para hacerlas efectivas* y *el interés y colaboración de los propios alumnos*. Las actividades a realizar pueden caer dentro de alguno de los siguientes grupos:

- *Seminarios impartidos sobre temas específicos.*
- *Trabajos voluntarios realizados por los alumnos.*
- *Conferencias invitadas.*
- *Workshop de trabajos realizados por los alumnos sobre temas de objetos.*

Desarrollo de las clases de teoría

Las clases de teoría se desarrollan de forma similar a las de la asignatura de Ingeniería del Software, esto es, utilizando una variante de la clase magistral, donde el profesor se

apoya en un retroproyector y en la pizarra para el desarrollo de los siete temas de los que se compone el temario.

Los alumnos cuentan de antemano con las transparencias de los temas para que no tengan que tomar apuntes en el sentido clásico del término, y puedan prestar atención a las explicaciones, completando las transparencias con las notas que cada uno crea oportuno. Además, permite que el alumno que lo desee intervenga en cualquier momento para hacer una pregunta o solventar una duda y no, como en el dictado de apuntes, para pedir que se repita una frase.

Evaluación de la parte teórica

La forma principal de evaluar la parte teórica de esta asignatura es mediante la realización de una prueba escrita. Aunque se puede eliminar una parte del mismo realizando y defendiendo un trabajo relacionado con la Orientación a Objetos.

En la Figura 5.8 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura.

Si (***Teoría*** ≥ 4,75) **y** (Práctica ≥ 5.0)
    Nota Final = (***Teoría****0,5) + (Práctica*0,5) + Nota trabajos
Sino
    ∅
Fin si

Figura 5.8. Influencia de la nota de la parte teórica en la nota final de Programación Orientada a Objetos

A la hora de elaborar el examen de la asignatura se tienen en cuenta los siguientes aspectos:

- El examen se divide en dos partes que hay que aprobar por separado para superar la prueba. La primera parte se centra en la evaluación de los conceptos básicos desarrollados en la asignatura, mientras que la segunda es un conjunto de supuestos prácticos y temas a desarrollar.
- La primera parte del examen está compuesta por un conjunto de cuestiones de respuesta abierta y corta, donde se prima evaluar la capacidad de asimilación, de comprensión y de relacionar los conceptos desarrollados en la asignatura, sobre la capacidad memorística del alumno.
- La segunda parte puede eliminarse realizando y defendiendo por parejas un trabajo sobre la tecnología de objetos.

Bibliografía básica de referencia

La lista de títulos que se les propone como bibliografía básica de consulta es:

- **Bishop, Judy**. "*Java. Fundamentos de Programación*". 2ª Edición. Addison-Wesley, 1999.
- **Deitel, H. M. and Deitel, P. J.** "*C++ How To Program*". 2nd Edition. Prentice Hall, 1998.
- **Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John.** "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.
- **Glass, Graham and Schuchert, Brett**. "*The STL <Primer>*". Prentice Hall, 1996.
- **Joyanes Aguilar, Luis.** "*Programación Orientada a Objetos*". 2ª Edición Osborne McGraw-Hill. 1998.
- **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.
- **OMG.** "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. [Última vez visitado, 14/2/2000]. June, 1999.
- **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.
- **Rumbaugh, J., Jacobson, I. and Booch, G.** "*The Unified Modeling Language Reference Manual*". Addison-Wesley, 1999.
- **Stroustrup, Bjarne.** "*El Lenguaje de Programación C++*". 3ª Edición Addison-Wesley, 1998.
- **SUN Microsystems.** "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

5.3.1.2 Desarrollo comentado del programa de teoría

En este epígrafe se va a desarrollar con mayor grado de detalle el programa de la parte teórica de la asignatura Programación Orientada a Objetos. Este programa se ha dividido en tres partes o unidades docentes.

En la primera unidad docente, **Conceptos básicos**, se parte de que el alumno ya ha sido introducido en la tecnología de objetos en la asignatura Ingeniería del Software cursada en el primer cuatrimestre. El objetivo de esta unidad docente se convierte, entonces, en introducir una serie de temas que serán útiles en el resto de la asignatura, porque se desea que el discente tenga una base conceptual de los mismos y constancia de su importancia; los temas a tratar son los lenguajes de programación y la reutilización del software.

La segunda unidad docente, **Diseño orientado a objetos básico**, se profundiza en el modelo objeto y sus implicaciones en el modelo de sistemas software. Se utiliza como ejemplos concretos los modelos objeto de C++ y Java para ilustrar las explicaciones.

La tercera y última unidad docente, **Diseño orientado a objetos avanzado**, introduce temas de mayor entidad en el diseño orientado a objetos, haciendo uso de las bases introducidas en la unidad docente dos.

Es obvio que, debido a los créditos asignados a la asignatura, no es posible extenderse en todas los temas por igual. En algunos casos la explicación se verá reducida a una breve referencia o comentario, mientras que en otros, que se consideran más representativos y de mayor actualidad, se profundiza más en las explicaciones.

Igual que en la descripción del temario de Ingeniería del Software, cada parte está dividida en una serie de temas, subtemas y epígrafes; donde todos los temas van a ser descritos siguiendo el patrón de documentación compuesto por las entradas: *título, descriptores, objetivos, contenido, resumen* y *bibliografía*.

Unidad Docente I: Conceptos básicos (4 Horas)

Tema 1. Lenguajes de programación orientados a objetos (2 Horas)

1.1 Historia y evolución (10 Minutos)

1.2 Clasificación de los LPOO (40 Minutos)

1.2.1 Introducción
1.2.2 Clasificación de Wegner
1.2.3 Clasificación de Tesler
1.2.4 Clasificación según el origen

1.3 Características de los LPOO (10 Minutos)

1.4 Revisión de algunos de los principales LPOO (1 Hora)

1.4.1 Simula
1.4.2 Smalltalk
1.4.3 C++
1.4.4 Eiffel
1.4.5 Java

Tema 2. Orientación a Objetos y reutilización del software (2 Horas)

2.1 Introducción (40 Minutos)

2.1.1 Aclaraciones previas
2.1.2 Conceptos básicos
2.1.3 Reutilización sistemática
2.1.4 Soporte a la reutilización

2.2 Ciclo de vida de la reutilización (30 Minutos)

2.2.1 Generalidades
2.2.2 Patrón general del proceso de reutilización
2.2.3 Actividades en el ciclo de vida de la reutilización

2.3 Reutilización y Orientación a Objetos (15 Minutos)

2.3.1 Generalidades
2.3.2 Dependencias de la OO con respecto a la reutilización

2.4 Reutilización en las fases del ciclo de vida (35 Minutos)

2.4.1 Análisis de dominio
2.4.2 Análisis de requisitos

2.4.3 Diseño

2.4.4 Implementación

Unidad Docente II: Diseño orientado a objetos básico (14 Horas)

Tema 3. Técnicas básicas de diseño orientado a objetos (8 Horas)

3.1 Clases y tipos (90 Minutos)

3.1.1 Concepto intuitivo de clase

3.1.2 Definición de clase

3.1.3 Clases y tipos

3.1.4 Clases y objetos en UML

3.1.5 Clases y objetos en C++

3.1.6 Clases y objetos en Java

3.1.7 Descubrimiento de clases

3.1.8 Diseño de clases

3.2 Métodos (30 Minutos)

3.2.1 Concepto intuitivo

3.2.2 Definición

3.2.3 Tipos de métodos

3.2.4 Métodos en C++

3.3 Herencia (2 Horas)

3.3.1 Introducción

3.3.2 Herencia múltiple

3.3.3 Notación en UML

3.3.4 Herencia en C++

3.3.5 Herencia en Java

3.3.6 Clasificación de la herencia

3.3.7 Uso adecuado de la herencia

3.4 Polimorfismo (1 Hora)

3.4.1 Introducción

3.4.2 Polimorfismo en C++

3.5 Asociaciones, agregaciones y composiciones (75 Minutos)

3.5.1 Introducción a la relación de asociación

3.5.2 Asociaciones en C++

3.5.3 Introducción a la relación de agregación

3.5.4 Agregaciones en C++

3.5.5 Introducción a la relación de composición

3.5.6 Composiciones en C++

3.6 Interfaces (30 Minutos)

3.6.1 Concepto de interfaz

3.6.2 Interfaces en Java

3.7 Módulos (15 Minutos)

3.7.1 Introducción

3.7.2 Modularidad en UML

3.7.3 Modularidad en C++

3.7.4 Modularidad en Java

3.8 Principios del diseño orientado a objetos (1 Hora)

3.8.1 El principio abierto/cerrado

3.8.2 El principio de sustitución de Liskov

3.8.3 El principio de inversión de dependencia

3.8.4 El principio de separación de la interfaz

3.8.5 El principio de equivalencia reutilización/revisión

3.8.6 El principio de dependencia acíclica

3.8.7 El principio de las dependencias estables

3.8.8 El principio de las abstracciones estables

Tema 4. Genericidad (4 Horas)

4.1 Introducción (20 Minutos)

4.1.1 Presentación de la genericidad

4.1.2 Doble dirección en la generalización de tipos

4.1.3 Coste de la genericidad

4.2 Tipos paramétricos y clases genéricas (20 Minutos)

4.2.1 Necesidad de la parametrización de tipos

4.2.2 Definiciones

4.3 Programación paramétrica (20 Minutos)

4.4 Genericidad en C++ (3 Horas)

4.4.1 Plantillas

4.4.2 STL

Tema 5. Manejo de excepciones (2 Horas)

5.1 Conceptos básicos (20 Minutos)

5.1.1 Introducción a las excepciones

5.1.2 Definiciones

5.1.3 Excepciones como objetos

5.2 Tratamiento de excepciones (30 Minutos)

5.2.1 Contraejemplos

5.2.2 Principios del tratamiento de excepciones

5.2.3 La cadena de llamadas

5.3 Mecanismo de excepciones (1 Hora)

5.3.1 Construcciones básicas

5.3.2 Clases de excepciones

5.4 Guías para el diseño de excepciones (10 Minutos)

Unidad Docente III: Diseño orientado a objetos avanzado (11 Horas)

Tema 6. Patrones de diseño (8 Horas)

6.1 Patrones software (1Hora)

6.1.1 Introducción

6.1.2 Origen e historia de los patrones

6.1.3 Concepto intuitivo de patrón

6.1.4 Definición de patrón software
6.1.5 Componentes esenciales de los patrones software
6.1.6 Descripción de los patrones software
6.1.7 Tipos de patrones software
6.1.8 Patrones y Orientación a Objetos

6.2 Patrones arquitectónicos (90 Minutos)

6.2.1 Introducción
6.2.2 Patrón Modelo-Vista-Controlador
6.2.3 Otros patrones arquitectónicos

6.3 Patrones de creación (90 Minutos)

6.3.1 Introducción
6.3.2 Patrón Factoría Abstracta
6.3.3 Otros patrones de creación

6.4 Patrones estructurales (90 Minutos)

6.4.1 Introducción
6.4.2 Patrón Composite
6.4.3 Otros patrones estructurales

6.5 Patrones de comportamiento (90 Minutos)

6.5.1 Introducción
6.5.2 Patrón Observador
6.5.3 Otros patrones de comportamiento

6.6 Antipatrones (1 Hora)

6.6.1 Introducción
6.6.2 Tipos de antipatrones
6.6.3 Ejemplos de antipatrones

Tema 7. Diseño por contrato (3 Horas)

7.1 Introducción al diseño por contrato (20 Minutos)

7.1.1 Introducción
7.1.2 Mecanismos básicos de fiabilidad
7.1.3 Comentarios sobre la corrección
7.1.4 Definición

7.2 Aserciones: Pre y postcondiciones (90 Minutos)

7.2.1 Expresión de una especificación
7.2.2 Aserciones
7.2.3 La filosofía de los contratos
7.2.4 Qué no son las aserciones
7.2.5 Un ejemplo explicativo

7.3 Invariantes de clase (30 Minutos)

7.3.1 Definición
7.3.2 Preservación de los invariantes
7.3.3 Invariantes y contratación
7.3.4 La corrección de una clase

7.4 Aserciones en C++ (40 Minutos)

Tabla 5.24. Estructura detallada del programa de teoría de Programación Orientada a Objetos

Unidad Docente I: Conceptos Básicos

Objetivo genérico

El objetivo de esta unidad docente es introducir al alumno en dos aspectos que van a ser claves en el desarrollo de la asignatura: *los lenguajes de programación orientados a objetos* y *la búsqueda de la reusabilidad del software desarrollado*.

Los lenguajes de programación no son el fin de esta asignatura, son el medio que se va a utilizar para que el alumno pueda concretar las nociones de diseño, centro de las unidades docentes dos y tres de esta asignatura.

La reutilización del software es uno de los mayores beneficios que tradicionalmente se ha asociado a la Orientación a Objetos. Debe hacerse hincapié en que la reutilización del software es algo independiente del paradigma de desarrollo adoptado, y que no se consigue sólo por adoptar la tecnología de objetos, aunque es cierto que la Orientación a Objetos ofrece facilidades que favorecen enormemente el desarrollo para y con reutilización.

Esta unidad supone el 13% de la asignatura, estando compuesta por dos temas, **Lenguajes de programación orientados a objetos** y **Orientación a Objetos y reutilización del software**, que se detallan a continuación.

Tema 1: ***Lenguajes de Programación Orientados a Objetos***

Descriptores

Programación Orientada a Objetos; lenguaje de programación orientado a objetos; lenguaje basado en objetos; lenguaje basado en clases; lenguaje orientado a objetos; comprobación de tipos; lenguaje orientado a objetos puro; lenguaje orientado a objetos híbrido; ligadura temprana; ligadura tardía; metadato; biblioteca de clases; entorno de desarrollo; corrutina; diseño por contrato.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir los conceptos básicos de los lenguajes de programación orientados a objetos.
- Presentar los orígenes históricos, así como la evolución e interrelación, de los lenguajes de programación orientados a objetos más relevantes.
- Diferenciar las características de los lenguajes de programación orientados a objetos puros frente a las propias de los lenguajes híbridos.
- Exponer las características principales que en un lenguaje de programación orientado a objetos se derivan del sistema de tipos que soporta.
- Enunciar los elementos propios y características deseables que se le pueden exigir a un lenguaje de programación orientado a objetos.
- Repasar someramente las características más notables de algunos de los lenguajes de programación más difundidos.

Contenidos

1.1 Historia y evolución
1.2 Clasificación de los LPOO
1.3 Características de los LPOO
1.4 Revisión de algunos de los principales LPOO

Tabla 5.25. Contenido del primer tema del programa teórico de Programación Orientada a Objetos

Resumen

Aunque el aprendizaje de uno o varios lenguajes de programación orientados a objetos no es el objetivo principal de esta asignatura, si que van a ser la principal herramienta con la que llevar a la práctica los conceptos impartidos. Así, se van a exponer los

principios fundamentales que, de forma genérica, deben conocer sobre estos lenguajes de programación; para ello este tema se ha divido en cuatro apartados principales.

En el primer apartado se presentan los orígenes y evolución de los lenguajes de programación orientados a objetos desde la entrada en escena a finales de 1966 del que se considera su primer representante, el Simula 67, hasta los lenguajes que mayor influencia están alcanzando en la actualidad.

El segundo apartado se utiliza para presentar diferentes clasificaciones de los múltiples lenguajes de programación. En concreto se presentan tres clasificaciones: *la clasificación de Peter Wegner*, *la clasificación de Tesler* y *la clasificación atendiendo al origen del lenguaje*.

Peter Wegner [Wegner, 1987] distingue tres tipos de lenguajes de programación según su vinculación con los conceptos de clase y objeto. Así se tienen:

- ***Lenguajes Basados en Objetos:*** Soportan objetos. Es decir, disponen de componentes caracterizados por un conjunto de operaciones (comportamiento) y un estado.
- ***Lenguajes Basados en Clases:*** Disponen, además de objetos, de componentes tipo clase con operaciones y estado común.
- ***Lenguajes Orientados al Objeto:*** Además de objetos y clases ofrecen mecanismos de herencia.

L. Tesler [Tesler, 1993] propone una clasificación basada en dos características: *la orientación del lenguaje* y *su carácter con respecto a la ligadura de los atributos a los nombres*. Se tienen así los cuatro tipos de lenguajes de programación orientados a objetos que se presentan en la Figura 5.9.

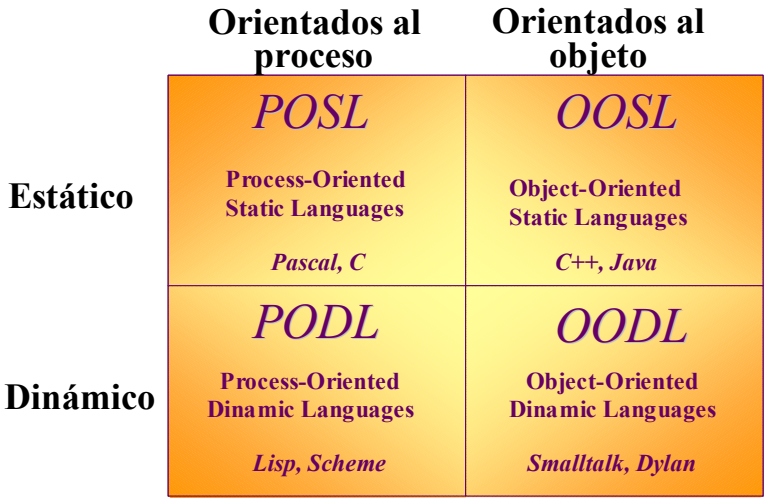

Figura 5.9. Clasificación de Tesler para los LPOO

Se aprovecha esta clasificación para introducir la vinculación de estos lenguajes con el sistema de tipos que soportan. Se define sistema de tipos como *las reglas que*

establecen la utilización de los tipos su corrección en un lenguaje de modelado o realización. De esta forma se introducen los conceptos de *tipado* y de *ligadura*.

El *tipado es el proceso de declarar cuál es el tipo de información que puede contener una variable* [Joyanes, 1998]. Se distinguen lenguajes estrictos y débiles en el chequeo de los tipos, incluso lenguajes sin tipo como Smalltalk.

La *ligadura es el proceso de asociar un atributo a un nombre. En el caso de las funciones, el término ligadura (binding) se refiere a la conexión o enlace entre una llamada a función y el código real ejecutado como resultado de la llamada* [Joyanes, 1998].

Los conceptos de tipos estrictos y tipos estáticos son completamente diferentes. La noción de tipos estrictos hace referencia a la consistencia de tipos, mientras que la asignación estática de tipos, también conocida como ligadura estática o ligadura temprana, se refiere a que los nombre se ligan con sus tipos en tiempo de compilación.

La ligadura dinámica o ligadura tardía significa que los tipos de las variables y expresiones no se conocen hasta el tiempo de ejecución.

Al ser la comprobación estricta de tipos y la ligadura dos conceptos independientes, un lenguaje puede tener comprobación estricta de tipos y tipos estáticos (Ada), puede tener comprobación estricta de tipos pero soportar enlace dinámico (Object Pascal o C++), o no tener tipos y admitir ligadura dinámica (Smalltalk).

Una tercera forma muy utilizada para clasificar los lenguajes de programación orientados a objetos es según su origen [Piattini, 1996], distinguiéndose entre lenguajes puros e híbridos.

Un *lenguaje de programación orientado a objetos puro* es un lenguaje diseñado para soportar únicamente el paradigma orientado al objeto, en el que todo gira entorno al concepto de objeto.

Un *lenguaje de programación orientado a objetos híbrido* es el que soporta otros paradigmas de programación (*estructurado, funcional…*) además del orientado al objeto. Los lenguajes híbridos se construyen a partir de otros lenguajes existentes, tales como C o Pascal, de los cuales se derivan. Es posible utilizar estos lenguajes de un modo no orientado al objeto, lo que suele ser fuente de críticas y disputas dentro de la comunidad de la Orientación a Objetos [Kölling, 1999a].

En el tercer apartado se enuncian y se explican someramente los elementos concretos que pueden aparecer en un lenguaje de programación orientado a objetos. También se mencionan las características que pueden servir para la elección de un determinado lenguaje de programación: *conceptos claros, orientado a objetos puro, seguridad, alto nivel, modelo de ejecución simple, sintaxis fácil de entender, eliminación de redundancias, pequeño, facilidad de transición a otros lenguajes,*

soporte para el aseguramiento de la corrección y *entorno de trabajo agradable* [Kölling, 1999a].

En el cuarto y último apartado se presentan las características más destacables de algunos lenguajes de programación orientados a objetos. En concreto se habla de Simula, Smalltalk, C++, Eiffel y Java.

Simula [SIS, 1987] se presenta por ser considerado el primer lenguaje orientado al objeto. Se aprovecha para introducir el concepto de corrutina, que aprovechaban los programadores de este lenguaje para implementar un pseudoparalelismo en sus programas.

Smalltalk [Goldberg and Robson, 1983] fue el primer lenguaje orientado a objetos puro. Ha sido uno de los lenguajes que más ha influido en la aparición de nuevos lenguajes, así como en muchos de los entornos gráficos que hoy se encuentran en los sistemas operativos más utilizados.

C++ [Stroustrup, 1997] es el lenguaje híbrido por excelencia, el que más controversias levanta dentro de la comunidad de la Orientación a Objetos y, actualmente, el más utilizado para abordar proyectos reales realizados bajo el paradigma objetual.

Eiffel [Meyer, 1992] es uno de los mejores lenguajes de programación orientados a objetos puros, aunque no ha tenido una difusión acorde a su calidad en los sectores industriales.

Java [SUN, 2000] es el lenguaje de moda actualmente, aunque todavía le falta madurar. Ha sido uno de los lenguajes que más rápidamente se han difundido y han sido aceptados internacionalmente. Su vinculación con Internet y su independencia de plataforma son sus mejores bazas para copar una buena parte de la parcela de los lenguajes de programación orientados a objetos.

Bibliografía

- ***Citada en las transparencias del tema:***

[Cardelli and Wegner, 1985] Cardelli, L. and Wegner, P. "*On Understanding Types, Data Abstraction and Polymorphism*". ACM Computing Surveys, 17(4). 1985.

[Goldberg and Robson, 1983] Goldberg, Adele and Robson, David. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Joyanes, 1998] Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Kölling, 1999] Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 1: Languages*". Journal of Object-Oriented Programming, 11(8):8-15. January, 1999.

[Meyer, 1992] Meyer, Bertrand. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

[SUN, 2000] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

[Tesler, 1981] Tesler, L. "*The Smalltalk Environment*". Byte, 6(8). August, 1981.

[Tesler, 1993] Tesler, L. "*Object-Oriented Dynamic Languages*". In Proceedings of the Object Expo Conference. July, 1993.

[Wegner, 1987] Wegner, Peter. "*The Object-Oriented Classification Paradigm in Research Directions on Object-Oriented Programming*". MIT Press, Cambridge, MA, 1987.

- ***Lecturas complementarias:***

"*The Real Stroustrup Interview*". IEEE Computer, 31(6):110-114. June, 1998.

> En 1998 se difundió por Internet una supuesta entrevista con Bjarne Stroustrup en la prácticamente el autor de C++ tiraba por los suelos su obra. Así en el mes de junio de este mismo año se publica en la revista IEEE Computer una entrevista, esta vez de verdad con Stroustrup, donde da su opinión sobre el papel de C++ en el mundo de la Programación Orientada a Objetos.

Interactive Software Engineering. "*Object-Oriented Languages: A Comparison*". **Interactive Software Engineering (ISE)**. http://www.eiffel.com/doc/manuals/technology/oo_comparison/index.html [Última vez visitado, 24-2-2000]. 1999.

> Tablas comparativas entre Eiffel, C++, Java y Smalltalk.

Joyner, Ian. "*C++?? A Critique of C++ and Programming and Language Trends of the 1990s*". 3rd edition http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/cppcv3.pdf. [Última vez visitado 7/1/2000]. 1996.

> Informe crítico sobre C++, comparando sus características con otros lenguajes de programación, en especial con Eiffel.

Kay, Alan C. "*The Early History of Smalltalk*". In Proceedings of the second ACM SIGPLAN conference on History of programming languages – HOPL II. (April 20 - 23, 1993, Cambridge United States). ACM SIGPLAN Notices, 28(3):69-75. March, 1993.

Artículo sobre los orígenes de Smalltalk.

Meyer, Bertrand. "*Approaches to Portability*". Journal of Object-Oriented Programming (JOOP), 11(4):68-70. July/August, 1998.

Comparativa de los bytecodes de Java con el método propio de Eiffel consistente en utilizar C como lenguaje intermedio para conseguir la portabilidad del software.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

En relación con el presente tema se destaca el capítulo 35, **De Simula a Java y más allá: los principales entornos y lenguajes O-O**, donde se repasan algunos de los lenguajes de programación orientados a objetos de mayor peso.

Piattini Velthuis, Mario Gerardo. "*Selección de Lenguajes de Programación Orientados al Objeto: ¿Cuestión de Religión?*". Revista BASE de la ALI (Asociación de Doctores, Licenciados e Ingenieros en Informática), Nº24:58-62. Abril, 1994.

Artículo intoductorio al mundo de los LPOO.

Prechelt, Lutz. "*Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences*". Communications of the ACM, 42(10):109-112. October, 1999.

Artículo en el que se presenta un experimento llevado a cabo para comparar la eficiencia de estos lenguajes de programación.

Rans, Michael. "*A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development*". http://users.ox.ac.uk/~ball0370/documents/oo.pdf [Última vez visitado, 22/12/1999]. November, 1999.

Resumen de la historia de algunos lenguajes de programación orientados a objetos destacados.

Stroustrup, Bjarne. "*The Design and Evolution of C++*". Addison-Wesley, 1994. Reprinted with corrections in April, 1995.

Libro del autor de C++ explicando la historia, diseño y evolución de C++.

- ***Referencias utilizadas para preparar las clases:***

Berard, Edward V. "*Object-Oriented Programming Languages*". The Object Agency. http://www.toa.com/pub/oopl_article.txt. [Última vez visitado, 24-2-2000]. 1989.

Buena introducción a las características de los lenguajes de programación orientados al objeto.

Budd, Timothy. "*Programación Orientada a Objetos*". Addison-Wesley Iberoamericana, 1994.

Referencia clásica en toda asignatura de Programación Orientada a Objetos, traducción al español de [Budd, 1991]. En su capítulo 20, **Información adicional**, se incluyen resúmenes introductorios de C++, Objective-C, Smalltalk y Object-Pascal. También se comentan algunas características para la elección de un LPOO.

Booch, Grady. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Dedica su apéndice A, **Lenguajes de programación orientados a objetos**, a introducir diversos LPOO: *Smalltalk, Object Pascal, C++, CLOS, Ada* y *Eiffel*.

Devis Botella, Ricardo. "*Smalltalk: Una Aproximación Práctica*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(6):16-24. Abril, 1995.

Una introducción a Smalltalk.

Graham, Ian. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Su capítulo 3, **Lenguajes de programación orientados a objetos y basados en objetos**, es una de las referencias más completas al tema hasta la fecha de su edición (1994).

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

En su capítulo 1, **El desarrollo de software**, tiene apartados donde presenta la evolución y la clasificación de los lenguajes de programación orientados a objetos.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

En su capítulo 1, **Language principles**, realiza un repaso por las características de los lenguajes de programación. Dentro de su segundo capítulo, **Entities and types**, dedica un apartado a la evolución de los lenguajes de programación orientados a objetos, desarrollándolo de una manera concisa, centrándose en los lenguajes que más han influido en los LPOO más influyentes en la actualidad.

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Incluye un apartado dedicado a los lenguajes de programación.

Rumbaugh, James, Blaha, **Michael, Premerlani, William, Eddy, Frederick, Lorensen, William**. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Su capítulo 15, **Lenguajes orientados a objetos**, presenta nociones de cómo pasar del diseño a la implementación, introduce las características que deben tener estos lenguajes y realiza un breve recorrido introductorio por Smalltalk, C++, Eiffel, CLOS y los lenguajes de programación de bases de datos.

Tema 2: *Orientación a Objetos y Reutilización del Software*

Descriptores

Reutilización del software; elemento reutilizable (*asset*); repositorio; componente; desarrollo para reutilización; desarrollo con reutilización; desarrollo basado en componentes; patrón; *framework*; tipo abstracto de datos.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar la importancia de la reutilización del software para el aumento de la productividad y de la calidad de los productos software desarrollados.
- Diferenciar la reutilización sistemática del software como una disciplina en sí misma, ortogonal a cualquier paradigma de desarrollo.
- Presentar la relación simbiótica existente en el binomio reutilización y Orientación al Objeto.
- Enumerar las diferentes facilidades que ofrece la tecnología de objetos a la reutilización del software.
- Introducir el ciclo de vida de la reutilización del software.

Contenidos

2.1 Introducción
2.2 Ciclo de vida de la reutilización
2.3 Reutilización y Orientación a Objetos
2.4 Reutilización en las fases del ciclo de vida

Tabla 5.26. Contenidos del tema dos del programa teórico de Programación Orientada a Objetos

Resumen

La reutilización del software es considerada por muchos autores como uno de los enfoques más adecuados para incrementar la productividad, ahorrar tiempo y reducir los costes de los desarrollos de software [Biggerstaff, 1992], [McClure, 1997].

El tema se organiza en cuatro apartados principales. En el primero de ellos se presentan las definiciones de los conceptos fundamentales de la disciplina de la reutilización sistemática del software.

El segundo apartado introduce las actividades propias del proceso de reutilización, distinguiendo aquéllas que son propias del desarrollo para reutilización de las propias del desarrollo con reutilización.

Los modelos de ciclo de vida del software tradicionales se han concebido bajo la premisa de que se iban a aplicar en el desarrollo de un producto software concreto, comenzando desde cero. Estos modelos generalmente se estructuran en una serie de fases que van desde los estudios de viabilidad hasta la implantación y mantenimiento del producto.

Los procesos que hacen uso de la reutilización dentro de un dominio de aplicación presentan una perspectiva diferente, de forma que el proceso software que soporta reutilización tiende a estructurarse en dos procesos distintos y separados: *Ingeniería de Dominio* o *Desarrollo Para Reutilización* e *Ingeniería de Aplicación* o *Desarrollo Con Reutilización* [Karlsson, 1995].

El motivo principal para contar con dos procesos separados es que cada uno de ellos tiene unos objetivos diferentes. Mientras que el desarrollo para reutilización se centra en el desarrollo de los elementos reutilizables, el desarrollo con reutilización se dirige hacia la construcción de un producto software individual, adaptando e integrando los elementos reutilizables existentes.

En el tercer apartado se relacionan las disciplinas de la Orientación a Objetos y de la reutilización del software.

La reutilización se cita de forma sistemática como uno de los principales objetivos de la tecnología de objetos (*junto con la reducción del tiempo de desarrollo, el aumento de calidad, la mejora de la productividad y la facilidad de mantenimiento de las aplicaciones [McClure, 1997]*).

Sin embargo, en contra de los que pueda parecer, la reutilización es una disciplina por sí misma, ortogonal a cualquier paradigma de desarrollo. Aunque no deja de ser cierto que el binomio reutilización-orientación al objeto da lugar a una relación simbiótica entre estas tecnologías. Por un lado la Orientación a Objetos aporta un modelo de referencia con una gran cantidad de elementos que favorecen la reutilización (*abstracción, encapsulamiento, jerarquías de herencia y composición, delegación...*). Por otro lado, la reutilización se convierte en un punto esencial para que, con el uso de la tecnología de objetos, se puedan conseguir el resto de los beneficios potenciales que ésta ofrece [García, 2000].

No obstante, esta relación simbiótica podría expresarse como una relación de dependencia de la Orientación a Objetos con respecto de la reutilización, afirmación que se justifica con las siguientes razones:

- Por el hecho de utilizar técnicas orientadas a objetos no se está reutilizando automáticamente [Griss, 1995]. Se necesita un esfuerzo extra para obtener la

potencia que el modelo objeto puede ofrecer en pro de la reutilización del software [Meyer, 1994], [García et al., 1997].

- Los proyectos de gran tamaño realizados con técnicas de objetos pueden volverse incontrolables si no se realizan sobre la base de la reutilización [McClure, 1996].
- La simple adopción de la tecnología de objetos, cuidando el desarrollo para reutilización, no garantiza una reutilización sistemática efectiva del software. Para lograr una reutilización a gran escala se deben unir factores tecnológicos (*métodos orientados a objetos con soporte para la reutilización*) con factores organizativos (*soporte de la dirección, cambios en la organización y cambios en el personal*).

El cuarto y último apartado repasa la reutilización a lo largo del ciclo de vida del software. Los elementos software reutilizables no tienen que limitarse al nivel de abstracción de implementación. De hecho el aumento de la potencia de la reutilización pasa por elevar el nivel de abstracción de los elementos reutilizables.

Se hace un repaso por la reutilización en el análisis de dominio, el análisis de requisitos, el diseño y la implementación.

Bibliografía

- ***Citada en las transparencias del tema:***

[Ader et al., 1990] Ader, M., Nierstrasz, O., McMahon, S., Muller, G. and Pröfrock, A.-K. "*The ITHACA Technology: A Landscape for Object-Oriented Application Development*". In Proceedings of ESPRIT'90 Conference. Kluwer Academic Publisher. 1990.

[Biggerstaff, 1989] Biggerstaff, T. J. "*Design Recovery for Maintenance and Reuse*". IEEE Computer, 22(7):36-49. July, 1989.

[Brown and Wallnau, 1998] Brown, A. W. and Wallnau, K. C. "*The Current State of CBSE*". IEEE Software, 15(5):37-46. September-October, 1998.

[Do Prado Leite et al., 1994] Do Prado Leite, J. C. S., Sant'Anna, M. and de Freitas, F. G. "*Draco-PUC: A Technology Assembly for Domain Oriented Software Development*". In Proceedings of the Third International Conference on Software Reusability ICSR-3. W. B. Frakes editor. (Rio de Janeiro, Brazil, 1-4 November 1994). Pages 102-109. IEEE Press, 1994.

[DoD, 1992] DoD. "*DoD Software Reuse Vision and Strategy*". Technical Report 1222-04-210/40, Department of Defense (DoD Software Software Reuse Initiative), Falls Church, VA. 1992.

[García, 2000] García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[Girardi, 1998] Girardi, M. Rosario. "*Main Approaches to Software Classification and Retrieval*". En las actas del curso *Ingeniería del Software y Reutilización*: *Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

[Griss, 1993] Griss, Martin L. "*Software Reuse: From Library to Factory*". IBM System Journal, 32(4):1-23. November, 1993.

[Griss and Wentzel, 1993] Griss, M. L. and Wentzel, K. D. "*Hybrid Reuse with Domain-Specific Kits*". In Paulin, J. and Tracz, W. editors, WISR'93: 6th Annual Workshop on Software Reuse. Summary and Working Group Reports. 1993.

[Karlsson, 1995] Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

[Kruchten et al., 1984] Kruchten, P., Schonberg, E. and Schwartz, J. T. "*Software Prototyping Using the SETL Programming Language*". IEEE Software, 1(4):66-75. 1984.

[Krueger, 1992] Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

[Marqués, 1998] Marqués Corral, José Manuel. "*Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación*". En las actas del curso Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

[McClure, 1997] McClure, Carma. "*Software Reuse Techniques: Adding Reuse to the System Development Process*". Prentice-Hall, 1997.

[McIlroy, 1976] McIlroy, Doug. "*Mass-Produced Software Components*". In Software Engineering Concepts and Techniques; 1968 NATO Conference on Software Engineering. J. M. Buxton, P. Naur and B. Randell editors. Pages 88-98. Van Nostrand Reinhold, 1976.

[NATO, 1992] NATO. "*NATO Standard for Management of a Reusable Software Component Library*". Volume 2 (of 3 Documents). NATO Communications and Information Systems Agency (NACISA). 1992.

[Neighbors, 1984] Neighbors, J. M. "*The Draco Approach to Constructing Software from Reusable Components*". IEEE Transactions on Software Engineering, SE-10(5):564-574. September, 1984.

[Pastor and Ramos, 1995] Pastor, Óscar and Ramos, Isidro. "*OASIS Version 2 (2.2): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*". Servicio de publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-95.788. 1995.

[Peterson, 1991] Peterson, S. A. "*Coming to Terms with Software Reuse Terminology: A Model-Based*". ACM Software Engineering Notes, 16(2):45-51. April, 1991.

[Prieto-Díaz, 1987] Prieto-Díaz, Rubén. "*Classification of Reusable Modules*". IEEE Software 4(1):6-16, January, 1987.

[STARS, 1993] STARS. "*STARS Conceptual Framework for Reuse Processes (CFRP)*". Technical Report STARS-VC-A018/001/00, STARS. Version 3.0 Vols I & II. 1993.

[Tracz, 1995] Tracz, Will. "*Confessions of a Used Program Salesman: Institutionalizing Software Reuse*". Addison-Wesley, 1995.

[Voas, 1998] Voas, J. M. "*The Challenges of Using COTS Software in Component-Based Development*". IEEE Computer, 31(6):44-45. June, 1998.

[Wallnau, 1992] Wallnau, K. C. "*Towards an Extended View of Reuse Libraries*". In Proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR-5), Palo Alto, California (USA). 1992.

- ***Lecturas complementarias:***

Adolph, Steve. "*Whatever Happened to Reuse?*". Software Development. http://www.sdmagazine.com/breakrm/features/s9911f3.shtml [Última vez visitado, 24-2-2000]. November, 1999.

Artículo crítico sobre los beneficios económicos de la verdadera reutilización del software en la tecnología de objetos.

DoD. "*Software Reuse Executive Primer*". Produced by DoD Software Reuse Initiative. April, 1996.

Sencilla introducción a la reutilización.

Fernández Sánchez, José Luis. "*Reusabilidad y Desarrollo Orientado a Objetos*". En las actas de las Segundas Jornadas sobre Tecnología de Objetos. Madrid, noviembre, 1996. http://www.ati.es/GRUP_TRABAJO/LATIGOO/OOp96/Ponen7/atio6p07.html. [Última vez visitado, 29-2-2000]. 1996.

Artículo sobre la contribución de la tecnología de objetos para conseguir diferentes niveles de reutilización.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

En su capítulo 4, **Aproximaciones a la reutilización**, B. Meyer realiza una introducción a la reutilización que, sin entrar en muchas profundidades, es adecuada para tomar conciencia de sus ventajas, problemas, inhibidores y su estrecha relación con la tecnología de objetos.

Meyer, Bertrand. "*The Significance of Components*". Software Development. http://www.sdmagazine.com/uml/beyondobjects/s9911bo1.shtml [Última vez visitado, 24-2-2000]. November, 1999.

Visión de Meyer sobre los componentes y su relación con la ocultación de la información.

Meyer, Bertrand. "*What to Compose*". Software Development. http://www.sdmagazine.com/uml/beyondobjects/s0003bo.shtml [Última vez visitado, 12-3-2000]. March, 2000.

Siguiendo como su serie de artículos sobre componentes, Meyer aborda el tema de qué y cómo componer.

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición. McGraw-Hill, 1998.

En su capítulo 26, **Reutilización del software**, se tiene una visión concreta que aborda diferentes campos de la reutilización del software.

- ***Referencias utilizadas para preparar las clases:***

Anaya de Paez, Raquel. "*Desarrollo de Componentes Reutilizables en el Marco de OASIS*". Tesis Doctoral. Universidad Politécnica de Valencia. 1999.

En su segundo capítulo, **Marco Conceptual**, se tiene un estado del arte de la reutilización del software.

Cybulski, Jacob L. "*Introduction to Software Reuse*". Technical Report TR 96/4, Department of Information Systems. University of Melbourne (Australia). July, 1996.

Buen texto introductorio a la reutilización del software, con un enfoque que engloba elementos reutilizables de todos los niveles de abstracción.

García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

En su primer capítulo, **Introducción**, se presenta un estado del arte de la reutilización sistemática del software.

García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre, 1997.

Informe técnico sobre las actividades a llevar a cabo para en el desarrollo de elementos reutilizables bajo el paradigma objetual.

Girardi, M. Rosario. "*Main Approaches to Software Classification and Retrieval*". En las actas del curso *Ingeniería del Software y Reutilización: Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

Excelente tratado sobre la clasificación y recuperación de los elementos reutilizables. Buena referencia a las actividades a llevar a cabo en el desarrollo para y con reutilización.

Jacobson, Ivar, Griss, Martín L. and Jonsson, Patrik. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley, 1997.

Libro imprescindible en el tema de la reutilización sistemática del software. Especialmente interesante porque el método que proponen abarca todo el ciclo de vida, con un enfoque muy ligado al modelado del negocio y a la Orientación a Objetos.

Johnson, Ralph E. and Russo, Vincent F. "*Reusing Object-Oriented Designs*". Technical Report UIUCDCS 91-1696, University of Illinois, 1991.

Informe técnico que presenta técnicas orientadas al objeto para potenciar la reutilización de las clases: *clases abstractas* y *frameworks*.

Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

Libro donde se desarrolla la metodología REBOOT (*REuse Based on Object-Oriented Techniques*). Este enfoque intenta aumentar la productividad y la calidad en los desarrollos software, promoviendo la reutilización sistemática. Se distinguen dos procesos en el ciclo de la reutilización del software: *el desarrollo para reutilización* y *el desarrollo con reutilización*.

Kinoshita, Shigeyuki (leader), Yoshikawa, Hiroshi (sub-leader), Matsumoto, Yoshihiro, Nakase, Masaharu, Mashiyama, Yohei, Tsunekawa, Ikuyo, Miki, Ryoji, Yakazi, Tomoo, Furukawa, Yohichiro and Ishikawa, Yuichi. "*State of the Art of Reuse in Object-Oriented Development*". Japan GUIDE/SHARE. http://www.guide.org/jgs/jgsoo1.htm. June, 1996.

Informe que relaciona la adopción, por parte de empresas japonesas, de la tecnología de objetos con la reutilización que se pretende conseguir.

Marqués Corral, José Manuel. "*Reutilización Sistemática del Software*". Notas de conferencia. Escuela Universitaria de Ingeniería Técnica en Informática de Gestión – Edificio Politécnico – Ourense, 27 de noviembre de 1998.

Introducción a la reutilización sistemática del software.

Marqués Corral, José Manuel. "*Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación*". En las actas del curso *Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

Excelente estado del arte de los repositorios y bibliotecas de reutilización.

Nierstrasz, Oscar, Gibbs, Simon and Tsichritzis, Dennis. "*Component-Oriented Software Development*". Communications of the ACM, 33(9):160-165. September, 1992.

Artículo introductorio al desarrollo basado en componentes.

Unidad Docente II: Diseño Orientado a Objetos Básico

Objetivo genérico

Esta unidad docente tiene el objetivo de profundizar en los conceptos básicos del modelo objeto que fueron introducidos en la asignatura de Ingeniería del Software.

El refinamiento que sufren los elementos que forman parte de los modelos en la Orientación al Objeto es el síntoma del cambio de nivel de abstracción o cambio de fase de los artefactos software construidos. Es por este motivo importante conocer perfectamente la semántica de los componentes del modelo objeto para refinar los modelos de análisis en modelos de diseño, y estar familiarizado con el soporte que ofrecen los lenguajes de programación para implementar los modelos de diseño en código.

Así, se van a considerar temas de modelado básico, el uso adecuado de los componentes del modelo objeto (*clases, objetos y relaciones, principalmente*), a la par que se introducen la genericidad y el uso adecuado de las excepciones.

Esta unidad docente supone el **47%** del programa teórico de la asignatura Programación Orientada Objetos, compuesta por tres temas: **Técnicas básicas de diseño orientado a objetos**; **Genericidad**; y **Manejo de excepciones** que se detallan a continuación.

Es importante destacar la relación existente entre los conceptos presentados en esta unidad docente y el programa práctico de la asignatura, donde se utilizará de forma práctica un lenguaje de programación orientado a objetos, C++ en este caso, para plasmar los contenidos desarrollados.

Tema 3: ***Técnicas Básicas de Diseño Orientado a Objetos***

Descriptores

Diseño orientado a objetos; clase; tipo; objeto; instancia; atributo; método; herencia simple; herencia múltiple; polimorfismo; asociación; mapping; relación de uso; agregación; composición; relación estructural; relación procedural; paquete; espacio de nombres; modularidad; interfaz; diseño para reutilización; extensión del software; relación tipo/subtipo.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T7, T9** y **P2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.
- Estudio y comprensión de los fundamentos del diseño de sistemas software.
- Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Comprender los elementos fundamentales del modelo objeto: *clases, herencia, polimorfismo, agregaciones, composiciones, asociaciones, clases paramétricas* y *excepciones*.
- Estudiar el soporte de los elementos anteriores en algunos lenguajes de programación orientados a objetos, fundamentalmente C++ y ocasionalmente Java.
- Construir programas orientados a objetos.

Contenidos

3.1 Clases y tipos
3.2 Métodos
3.3 Herencia
3.4 Polimorfismo
3.5 Asociaciones, agregaciones y composiciones
3.6 Interfaces
3.7 Módulos
3.8 Principios del diseño orientado a objetos

Tabla 5.27. Contenidos del tema 3 del programa teórico de Programación Orientada a Objetos

Resumen

El diseño y posterior implementación de un sistema software bajo el paradigma orientado a objetos pasa por conocer las bases fundamentales con las que llevar a cabo

las construcciones software, los elementos del modelo objeto y su soporte en algunos lenguajes de programación orientados a objetos.

Este tercer tema hace un repaso por los elementos fundamentales del modelo objeto, pero con un enfoque más centrado en el diseño y en la implementación. Para ello se ha dividido en ocho apartados principales

El primer apartado se dedica a la entidad estructural por excelencia de los lenguajes orientados a objetos, la clase. Una clase es una descripción de objetos capaz de servir como molde para crear instancias o, lo que es lo mismo, objetos reales descritos por la clase. Este proceso de creación se conoce usualmente como instanciación. De esta forma una clase dicta la estructura y comportamiento de sus instancias (*objetos*), mientras que las instancias contienen localmente datos que se corresponden con la estructura dictada por la clase y que representan el estado del objeto.

Así pues, una clase puede definirse como *una construcción lingüística en un lenguaje orientado a objetos. Las clases implementan tipos y son plantillas a partir de las cuales se crean objetos. Los objetos de la misma clase tienen estructura y operaciones comunes de acuerdo a la definición de la clase* [Crespo, 2000].

Según la definición anterior, las clases implementan tipos y por tanto ayudan a la clasificación al nivel de un lenguaje orientado a objetos [Liskov, 1987]. Una clase es una plantilla a partir de la cual se crean objetos. Los objetos de la misma clase tienen estructura y operaciones comunes, descritas en su clase, y por tanto comportamiento uniforme. La estructura está dada por sus atributos y las operaciones por sus métodos, cada uno con su correspondiente signatura. Puestos en conjunto, los atributos y los métodos de una clase suelen recibir el nombre de **recursos** o **propiedades de la clase**.

Las clases tienen una o más interfaces que indican las operaciones que, a través de su correspondiente interfaz, son accesibles a los clientes que utilicen los objetos que dicha clase describe.

La relación entre el concepto de clase y el concepto de tipo es algo que suscita controversia. En muchos casos, una clase se corresponde directamente con un tipo, pero esto no siempre se cumple. En C++ una clase es un tipo definido por el usuario. En Java, un tipo puede ser un *tipo primitivo* o *una referencia a un tipo basada en una clase, una interfaz o una matriz*. Eiffel no equipara los dos conceptos; cada tipo debe estar basado en una clase. En Eiffel todos los tipos, incluso los primitivos y las matrices, se derivan de clases. Mientras que un usuario puede definir tipos como clases, los dos conceptos no son equivalentes porque se pueden derivar múltiples tipos de una clase mediante la genericidad [Joyner, 1999].

A partir de una clase se puede obtener el tipo que dicha clase implementa. Esto se puede hacer con relativa facilidad si el modelo de tipos es basado en estructura y signatura [Crespo, 2000]. Las clases siempre implementan tipos implícita o explícitamente. Por este motivo muchas veces en la bibliografía se dice que la clase es

una forma especial de tipo. Esto significa que para toda clase existe una especificación de tipo que caracteriza al conjunto de las instancias potenciales de la clase. La extensión de dicho tipo[25] se correspondería con el conjunto de instancias potenciales de la clase.

En general, una clase puede tener parámetros formales que se usan para definir módulos paramétricos o genéricos. Estas clases reciben el nombre de clases genéricas o paramétricas. En este caso una clase paramétrica es la implementación de una familia de tipos.

Concretando, una clase, en su rol de plantilla para crear instancias, suele denominarse en la bibliografía *tipo de objeto*; aunque, en rigor, una clase es la implementación de un tipo. Desde el punto de vista de los programadores, los tipos y las clases son elementos diferentes, debido a que la información del tipo sólo ofrece la especificación de un objeto[26]. Las clases describen especificaciones que pueden ser compartidas entre colecciones de objetos u otras clases, no sólo entre objetos con una única entidad, o instancias. Sin embargo, desde el punto de vista de los analistas, las clases y los tipos abstractos de datos son, en efecto, lo mismo [Graham, 1994].

Como conclusión, en la programación orientada a objetos un tipo y una clase no son exactamente lo mismo, considerándose a los tipos como *la puesta en vigor de la clase de los objetos, de modo que los objetos de distinto tipo no pueden intercambiarse o, como mucho, pueden hacerlo sólo de formas muy restringidas* [Booch, 1994].

Por otra parte, una clase no es únicamente la implementación de uno o más tipos, puede ser también, dependiendo del sistema orientado a objetos, lo que se suele denominar un *ciudadano de primera clase*. Por ejemplo, en Smalltalk o en OASIS [Letelier et al., 1998], [Carsí, 1999] una clase es también un metaobjeto instancia de una metaclase, y por tanto *ciudadano de primera clase*. En Java se baja un grado en este sentido ya que las clases dejan de ser *ciudadanos de primera clase*, pero pueden tratarse como un pseudoobjeto al que es posible consultarle su estado (*qué métodos tiene definidos la clase...*). También en Java una clase puede tener variables de clase, accesibles tanto para modificar como para consultar; de este modo podría decirse que esas variables constituyen una parte del estado de la clase vista como pseudoobjeto. Esa parte del estado es la única que puede modificarse dinámicamente. En C++, disminuyendo aún más el grado, se tiene que una clase sólo puede ser vista como un pseudoobjeto considerando el aspecto anterior, esto es, que puede tener variables de clase. En Eiffel no se puede ver a una clase de otra forma que no sea una plantilla que implementa tipos y describe objetos.

Se termina el apartado con unas nociones básicas sobre el diseño de clases, que van desde su identificación (haciendo especial hincapié en las tarjetas CRC - *Class,*

[25] La extensión de un tipo está formada por todos los objetos que se clasifican en dicho tipo.

[26] Un tipo puede definirse como *una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades* [Zilles, 1984].

Responsability and Collaboration - Clases, Responsabilidades, y Colaboraciones [Beck and Cunningham, 1989], [Wirfs-Brock, et al., 1990]), los diferentes tipos de clases que se pueden manejar (datos o manejadores de datos; pozos o fuentes de datos; vistas; auxiliares [Budd, 1991]) y una serie de heurísticas y errores frecuentes que se cometen en su diseño.

El segundo apartado del tema está dedicado a los métodos de las clases. En la mayoría de los lenguajes de programación orientados a objetos, las operaciones que los clientes pueden realizar sobre un objeto suelen declararse como métodos, que forman parte de la declaración de la clase. C++ utiliza el término *función miembro* para denotar el mismo concepto.

Una operación denota un servicio que una clase ofrecer a sus clientes. Se pueden distinguir cinco tipos de operaciones básicas, a saber [Booch, 1994]:

- ***Modificador:*** Una operación que altera el estado de un objeto.
- ***Selector:*** Una operación que accede al estado de un objeto, pero no altera ese estado.
- ***Iterador:*** Una operación que permite acceder a todas las partes de un objeto en algún orden perfectamente establecido.
- ***Constructor:*** Una operación que crea un objeto y/o inicia su estado.
- ***Destructor:*** Una operación que libera el estado de un objeto y/o destruye el propio objeto.

S. Lippman sugiere otra clasificación ligeramente diferente: *funciones de manejo, funciones de implantación, funciones de asistencia (todo tipo de modificadores) y funciones de acceso (equivalentes a los selectores)* [Lippman, 1989].

El apartado tercer se dedica a la herencia, uno de los baluartes de la programación orientada a objetos.

La noción de herencia tiene sus raíces en el estudio de la inteligencia artificial, donde las redes semánticas [Quillian, 1967] y los marcos [Minsky, 1981] son técnicas para representar el conocimiento acerca de los objetos y los conceptos estereotípicos, de forma que la relación entre conceptos más generales y más especializados se gestiona a través de la herencia de propiedades y procedimientos.

La relación de herencia en la programación orientada a objetos tiene un significado análogo a su sentido usual que se hace familiar a todo el mundo. Heredar es recibir características de otro, normalmente como resultado de alguna relación especial entre el que da y el que recibe [Danforth and Tomlinson, 1988]. La herencia es la adquisición de propiedades de generaciones anteriores.

En el contexto de la Orientación a Objetos, la herencia permite a nuevas descripciones de objetos basarse en descripciones existentes. Cuando una nueva clase de

objetos se define a partir de otras mediante herencia, sólo se necesita declarar de forma explícita las propiedades en las que difiere de las clases existentes, tomando el resto de éstas automáticamente.

La herencia es una relación entre clases en la que una clase comparte la estructura y/o el comportamiento definidos en una (*herencia simple*) o más clases (*herencia múltiple*) [Booch, 1994].

Los términos **hijo** y **padre**, **derivada** y **base**, **subclase** y **superclase** se utilizan para nombrar a una clase que hereda de otra y a esta última, respectivamente. La herencia es transitiva [Marqués, 1995], [Taivalsaari, 1996] y se utiliza la palabra **descendientes** para referirse a los hijos de una clase y a todos aquéllos que se encuentran en relación transitiva de herencia con los hijos. Por su parte, se emplea la palabra **ancestros** para referirse a los padres de una clase y a todos aquéllos de los cuales estos padres heredan transitivamente.

En los lenguajes orientados a objetos, las clases implementan tipos o familias de tipos y hay herencia entre clases. Se distinguen dos formas fundamentales de herencia de clases atendiendo a su uso: *para construir jerarquías de implementación* y *para implementar jerarquías de subtipos*.

Estas formas de utilización de la herencia son denominadas en [Sakkinen, 1989] **herencia incidental** y **herencia esencial** respectivamente.

La herencia esencial se corresponde con implementar relaciones de subtipo a través de la herencia, lo que se trata ampliamente en [Liskov, 1987]. La herencia incidental, también llamada herencia de reutilización o de compartición de código, se trata en [Snyder, 1991] y en [Taivalsaari, 1996]. El uso de la herencia por razones de implementación o reutilización se justifica sobradamente en muchos casos, aunque hay que tener en cuenta que también es posible cambiar la herencia de reutilización por una relación cliente apoyándose en delegación [Gamma et al., 1995].

Existen otras clasificaciones de la herencia. Así en [Ancona et al., 1992] se clasifica la herencia de acuerdo al grado de libertad para redefinir el cuerpo de los métodos heredados, distinguiéndose entre *herencia minimal, herencia regular* y *herencia conservadora*.

La herencia minimal permite redefinir los métodos con plena libertad, caso de C++ por ejemplo. La herencia regular requiere que se preserven algunas características semánticas de los métodos heredados, esto es así en Eiffel donde se imponen reglas relacionadas con las precondiciones y postcondiciones de los métodos. En la herencia conservadora los métodos se redefinen de forma que vistos desde el punto de vista del padre se comporten como antes.

En [Halbert and O'Brien, 1987] se habla de *herencia de implementaciones completas* y *herencia de implementaciones parciales*.

La herencia de implementaciones parciales se consigue gracias a la posibilidad de definir clases abstractas. Una **clase abstracta** es un caso especial de clase porque es una descripción incompleta de objetos, no pudiendo servir como plantilla para la construcción de instancias. En los sistemas orientados a objetos una clase se considera abstracta cuando contiene al menos un método para el cual no se ha proporcionado una implementación, denominándose método abstracto o diferido. Una clase puede ser completamente abstracta o tener cierto grado de concreción, comprometiéndose con una implementación particular. En el caso de C++, Java o Eiffel se pueden tener clases abstractas con cualquier nivel de concreción.

La herencia de implementaciones parciales consiste en definir clases abstractas y heredar de ellas para ir progresivamente completando la implementación. Por el contrario, la herencia de implementaciones completas consiste en heredar de una clase no abstracta para redefinir, añadir...

Por su parte, **Bertrand Meyer** define doce usos válidos de herencia agrupados en tres familias, llamadas *herencia de modelo, herencia de variación* y *herencia de software* [Meyer, 1997]. En la primera familia incluye lo que denomina herencia de subtipo, herencia de vista, herencia de restricción y herencia de extensión. En la segunda incluye herencia por variación funcional, herencia por variación de tipo y herencia por conversión diferida (esto es a abstracta). En la última familia incluye herencia de materialización, de estructura, de implementación y de facilidad que agrupa dos usos especiales, herencia de constantes y herencia de máquina. Casi todos estos usos podrían clasificarse como formas de herencia esencial o incidental. De hecho el propio autor indica que el segundo grupo, *herencia de variación*, no es ortogonal a los otros dos.

En este apartado también se ponen de manifiesto los problemas que se derivan de la herencia múltiple y cómo se soluciona en algunos lenguajes que la soportan, caso de C++ o Eiffel, haciendo hincapié en que no todos los lenguajes de programación orientados a objetos la soportan, como es el caso de Java o Smalltalk.

En el cuarto apartado se aborda el polimorfismo, concretamente el polimorfismo universal de inclusión que se deriva de la relación de herencia.

El polimorfismo[27] es un concepto de la teoría de tipos en el que un nombre puede denotar instancias de muchas clases diferentes en tanto en cuanto estén relacionadas por alguna superclase común. Cualquier objeto denotado por este nombre es capaz de responder a algún conjunto de operaciones de diversas formas [Booch, 1994].

Como ponen de manifiesto **Cardelli y Wegner** [Cardelli and Wegner, 1985], los lenguajes convencionales con tipos, como Pascal, se basan en la idea de que las funciones, los procedimientos y, por tanto, los operandos, tienen un único tipo. Tales

[27] El término polimorfismo viene del griego *poly*, que significa muchas, y *morfos*, que significa formas.

lenguajes se dice que son monomórficos, en el sentido de que todo valor y variable puede interpretarse que tiene un único tipo. Los lenguajes de programación monomórficos pueden contrastarse con los lenguajes polimórficos en los que algunos valores y variables pueden tener más de un tipo.

El concepto de polimorfismo fue descrito en primer lugar por **Strachey**, que habla de un polimorfismo *ad-hoc*, por el que símbolos como "+" pueden significar cosas distintas, lo que se denomina sobrecarga de operadores, aunque la coerción también es considerada por algunos autores como un tipo de polimorfismo ad-hoc, donde una operación separada semánticamente se relaciona también con las operaciones aritméticas. La coerción se manifiesta cuando el valor de un tipo se convierte en otro tipo diferente [Budd, 1991].

Hoy en día los lenguajes modernos admiten lo que se conoce como polimorfismo universal, distinguiéndose el polimorfismo paramétrico y el polimorfismo de inclusión. El polimorfismo paramétrico se refiere a la posibilidad de sustituir argumentos de un rango de tipos en una llamada a función, mientras que el polimorfismo de inclusión o de subclases se produce cuando un servicio definido en una clase se redefine en alguna de sus subclases manteniendo la misma signatura. Así, un mensaje enviado a un objeto instancia de esta clase o de cualquiera de sus subclases puede invocar cualquiera de estos servicios, según sea la clase a la que pertenezca el objeto que lo recibe.

El polimorfismo más interesante es el polimorfismo de inclusión, el cual va de la mano de la ligadura tardía, de forma que la ligadura de un método con un nombre no se determina hasta la ejecución. En C++, el desarrollador puede controlar si una función miembro de una clase utiliza ligadura temprana o tardía. Concretamente, si el método se declara como virtual se emplea ligadura tardía, y la función se considera polimórfica. Si se omite esta declaración virtual, el método utiliza ligadura temprana, resolviéndose la referencia en tiempo de compilación.

En el quinto apartado se discute cómo utilizar e implementar otras relaciones entre clases: *asociaciones, relaciones de uso* y *relaciones todo/parte*.

Las asociaciones constituyen el tipo más general de relaciones entre clases, a la vez que son las construcciones de con mayor debilidad semántica. La identificación de asociaciones entre clases es frecuentemente una actividad de análisis y de diseño preliminar, momento en el cual se comienza a describir las dependencias generales entre las abstracciones. A medida que se continúa el diseño y la implementación pueden refinarse, orientándolas hacia otras relaciones de clase más concretas.

En el diseño debe formularse una estrategia para implementar las asociaciones, para lo cual debe analizarse la forma en que serán utilizadas. Numerosas corrientes de modelado consideran a las asociaciones inherentemente bidireccionales [Rumbaugh et al., 1991], [Booch, 1994], [OMG, 1999], lo que es cierto en sentido abstracto. Pero si

las asociaciones sólo se van a recorrer en una dirección, se puede simplificar su implementación.

Si una asociación sólo se recorre en una dirección, es posible implementarla mediante un atributo que contenga una referencia a un objeto, lo que **Rumbaugh** denomina punteros ocultos o enterrados [Rumbaugh et al., 1991]. Si la multiplicidad es de "uno" se necesita una única referencia; si la multiplicidad es "muchos" se requiere un conjunto de referencias. Si el extremo "muchos" está ordenado se puede utilizar una lista en lugar de un conjunto. Una asociación cualificada con multiplicidad "uno" se puede implementar en forma de objeto diccionario. Un diccionario es un conjunto de pares de valores que hacen corresponder valores del selector con valores del destino.

Cuando las asociaciones se pueden recorrer en ambas direcciones, se pueden seguir tres aproximaciones para su implementación [Rumbaugh et al., 1991]:

- Se implementan como un atributo en una dirección sólo, y se hace una búsqueda cuando se requiere un recorrido en la otra dirección. Esta aproximación sólo es útil si hay una gran disparidad entre las frecuencias de recorrido en los dos sentidos, y si además es importante minimizar el coste de almacenamiento y el de actualización.
- Se implementan como atributos en ambas direcciones, utilizando referencias. Esto permite un acceso rápido, pero si se actualiza alguno de los atributos también el otro atributo deberá actualizarse para mantener la congruencia del enlace, lo que conduce a una ruptura de la encapsulación. Esta aproximación es útil cuando el número de accesos supera al de las actualizaciones.
- Se implementan como un objeto de asociación por separado, independiente de ambas clases [Rumbaugh, 1987]. Un objeto de asociación es un conjunto de parejas de objetos asociados que se almacenan en un único objeto de tamaño variable. Por eficiencia, un objeto de asociación se puede implementar empleando dos objetos de diccionario, uno para cada sentido.

En [Graham et al., 1997] se hace una fuerte crítica a las asociaciones bidireccionales, porque violan la encapsulación y comprometen la reutilización de las clases. Se propone la utilización de asociaciones unidireccionales, *mappings* al estilo de MOSES [Henderson-Sellers and Edwards, 1994] o SOMA [Graham, 1995] y que dieron lugar a las asociaciones de OML [Firesmith et al., 1998], que preserven la encapsulación, integrando en el objeto las reglas de negocio.

Mientras que una asociación denota una conexión semántica bidireccional, una relación *de uso* es un posible refinamiento de una asociación, por el que se establece qué abstracción es el cliente y qué abstracción es el servidor que proporciona ciertos servicios [Booch, 1994].

Las relaciones de uso equivalen a lo que **James Rumbaugh** denomina dependencias procedurales entre clase, proponiendo modelarlas como dependencias en los diagramas de clases de UML [Rumbaugh, 1998].

Las relaciones todo/parte van a estar representadas por las agregaciones y por las composiciones, tal y como se definen en UML 1.x [OMG, 1999], equivaliendo a la agregación por referencia y por valor respectivamente, según [Booch, 1994].

Una clasificación más rica en cuanto a las relaciones todo/parte se tiene en OML [Henderson-Sellers, 1997].

Una relación todo/parte es una forma fuertemente acoplada de asociación, con una cierta cantidad de semántica adicional. Son relaciones transitivas, esto es, si A es parte de B y B es parte de C, entonces A es parte de C. Además, son relaciones antisimétricas, si A es parte de B, entonces B no es parte de A.

En OMT [Rumbaugh et al., 1991], el método de Booch [Booch, 1994] y UML 1.x [OMG, 1999] las relaciones todo/parte son bidireccionales, frente al enfoque de OML donde son unidireccionales [Firesmith et al., 1998].

En UML 1.x se distingue entre agregación y composición.

La agregación es una relación más relajada, donde los tiempos de vida del objeto *todo* y de los objetos *partes* ya no están tan estrechamente ligados: se pueden crear y destruir instancias de cada clase involucrada en la relación de forma independiente. Es más, puesto que es posible que las *partes* sean compartidas estructuralmente, hay que decidir alguna política por la que su espacio de almacenamiento sea correctamente creado y destruido por sólo uno de los agentes que comporten referencia a cada una de las *partes*.

La composición indica una contención física, de manera que el objeto compuesto (*todo*) contiene físicamente a cada uno de sus objetos componentes (*partes*). El tiempo de vida de los objetos componentes está íntimamente ligado, cuando se crea una instancia del compuesto se crea una instancia de cada componente, cuando se destruye la instancia del compuesto se destruyen las instancias de cada componente.

El sexto apartado se dedica a las interfaces. Una interfaz puede definirse como un conjunto de operaciones referenciado por un nombre y que caracteriza el comportamiento de un elemento [OMG, 1999].

Las interfaces son una forma de declarar un tipo que se compone sólo de métodos abstractos, posibilitando que se escriba cualquier implementación para estos métodos. Una interfaz es una expresión de diseño puro, mientras que una clase es una mezcla de diseño e implementación [Arnold and Gosling, 1997].

Java incluye un mecanismo sintáctico para la definición de interfaces, de forma que introducen nuevos tipos de referencia cuyos miembros son sólo constantes y métodos abstractos. Este lenguaje admite que una interfaz extienda más de una interfaz (*herencia*

múltiple de interfaces) y que una clase implemente más de una interfaz, aspectos que contrastan con el hecho de que la herencia simple sea el único tipo de herencia permitido entre clases.

De esta manera Java presenta una aproximación al hecho de que la interfaz y la implementación son físicamente cosas distintas. Aunque si se restringen los diseños a la herencia simple, no existe una verdadera necesidad del uso de las interfaces en Java, porque cada clase implícitamente tiene una interfaz, sin la necesidad de declarar una *interfaz* explícita [Joyner, 1999], cumpliéndose la afirmación de que cada clase presenta una interfaz a sus usuarios [Stroustrup, 1997].

El siguiente apartado, el número siete, está dedicado a la modularidad en la Orientación a Objetos, pudiéndose considerar una extensión del apartado anterior.

Se discuten dos temas fundamentales, y muy relacionados entre sí:

- El agrupamiento lógico y físico de las clases, y
- La distinción entre interfaz e implementación.

El módulo lógico básico en la mayoría de los sistemas orientados a objetos es la clase, aunque para la mejor gestión y reutilización de estas clases deben organizarse en elementos de mayor grano, como se apunta desde la bibliografía especializada. Así, **Grady Booch** habla de categorías [Booch, 1994], **Bertrand Meyer** de *clusters* [Meyer, 1992], **Bjarne Stroustrup** de componente y posteriormente de espacios de nombres [Stroustrup, 1997], en Java se sigue una organización de clases en paquetes [Arnold and Gosling, 1997], al igual que en UML 1.x [OMG, 1999].

Físicamente, los lenguajes más extendidos organizan sus clases en ficheros, y éstos a su vez en directorios, como es el caso de Java.

Hasta aquí las semejanzas, derivadas todas ellas por el concepto recurrente de clase, presente en todos los lenguajes de programación orientados a objetos de una manera similar. Sin embargo, los lenguajes difieren en gran medida a la hora de organizar las clases en módulos físicos.

C++ hereda de C su organización de módulos, esto es, se suelen colocar las interfaces de los módulos en archivos cabecera (.h), mientras que las implementaciones de los módulos se sitúan en archivos de implementación (.cpp, .cc). Las dependencias entre módulos se declaran mediante directivas de inclusión. Además, se tiene el concepto de espacio de nombre para ofrecer un mayor control sobre los módulos.

En Eiffel la unidad básica de modularidad es la clase, ofreciendo clusters para su empaquetamiento, aunque se considera que este empaquetamiento de clases es cometido del entorno de desarrollo, no del lenguaje.

Java se asemeja en cierta forma a Eiffel gracias a sus paquetes, *packages*, aunque se diferencia de Eiffel en que los paquetes están definidos como parte del lenguaje y los

clusters de Eiffel no. Además, Java ofrece un esquema de cómo los nombres de las clases y los paquetes se corresponden con el sistema de ficheros.

En java la declaración de una clase y su implementación se almacenan en el mismo lugar y no se definen separadamente. Esto provoca que a veces los ficheros fuente sean muy grandes dado que cualquier clase debe estar definida por completo en un único archivo fuente.

El último apartado de este tema recoge algunos de los principios básicos del diseño orientado a objetos, estrechamente relacionados con el desarrollo para reutilización. Los principios expuestos se enuncian en [Martin, 1997a] y se detallan con mayor grado de detalle en [García et al., 1997]. En concreto los principios de diseño que se consideran son:

1. *El principio de abierto/cerrado.*
2. *El principio de sustitución de Liskov.*
3. *El principio de inversión de dependencia.*
4. *El principio de separación de la interfaz.*
5. *El principio de equivalencia reutilización/revisión.*
6. *El principio de cierre común.*
7. *El principio de reutilización común.*
8. *El principio de dependencia acíclica.*
9. *El principio de las dependencias estables.*
10. *El principio de las abstracciones estables.*

El ***principio abierto/cerrado*** establece que *las entidades software (clases, módulos, funciones...) deben estar abiertas para su extensión, pero cerradas para su modificación* [Meyer, 1988].

La naturaleza, simultáneamente abierta y cerrada de los sistemas orientados al objeto da soporte a la facilidad de mantenimiento, de reutilización y de verificación. Así, un sistema reutilizable debe estar abierto, en el sentido de que tiene que ser fácil de extender, y cerrado en el sentido de que debe estar listo para ser utilizados [Graham, 1994].

Cuando un único cambio en un programa produce una cascada de cambios en los módulos dependientes, el programa exhibe unos atributos no deseables debidos a un mal diseño. Así, el programa se convierte en un programa frágil, rígido, impredecible y en consecuencia **NO REUTILIZABLE**. Este principio ataca este vicio de forma directa, expresando la idea de que nunca se debe cambiar el diseño de los módulos. Cuando cambien los requisitos, se extiende el comportamiento de los módulos añadiendo nuevo código, pero nunca cambiando el código que ya funciona [Martin,1996a].

En el ***principio de sustitución de Liskov*** se busca la siguiente propiedad de sustitución: *si para cada objeto o_1 de tipo S hay un objeto o_2 de tipo T tal que para todos los programas P definidos en términos de T, el comportamiento de P no cambia cuando o_1 es sustituido por o_2, entonces S es un subtipo de T* [Liskov, 1987].

El principio de sustitución de Liskov puede entenderse de la siguiente forma: "*Las funciones que usan punteros o referencias a clases base deben ser capaces de utilizar objetos de las clases derivadas sin tener conocimiento de ello*" [Martin, 1996b], o también como que "*Las clases derivadas deben ser utilizables a través de la interfaz de la clase base, sin necesidad de que el usuario conozca la diferencia*" [Martin, 1997a].

El ***principio de inversión de dependencia*** puede enunciarse como sigue: *los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de las abstracciones.* Asimismo, *las abstracciones no deben depender de los detalles. Los detalles deben depender de las abstracciones* [Martin, 1996c].

El ***principio de separación de la interfaz*** se enuncia como sigue: *los clientes no deben ser forzados a depender de interfaces que no utilizan* [Martin, 1996d].

El ***principio de equivalencia reutilización/revisión***, expresa que: *la granularidad de la reutilización es la granularidad de la revisión. Sólo los componentes que son revisados a través de un sistema de distribución pueden ser reutilizados de forma efectiva. Este grano es el paquete* [Martin, 1996e].

El ***principio de cierre común*** establece que: *las clases en un paquete deben estar cerradas juntas frente a los mismos tipos de cambios. Un cambio que afecta a un paquete afecta a todas las clases del paquete* [Martin, 1996e].

El ***principio de reutilización común*** se enuncia como sigue: *las clases que pertenecen a un paquete se reutilizan juntas. Si se reutiliza una de las clases del paquete, se reutilizan todas* [Martin, 1996e].

El ***principio de dependencia acíclica*** establece que: la estructura de dependencia entre los paquetes debe ser un grafo dirigido acíclico (DAG). Esto es, no debe haber ciclos en la estructura de dependencia [Martin, 1996e].

El ***principio de las dependencias estables*** se enuncia como sigue: *Las dependencias entre paquetes en un diseño debe hacerse buscando la dirección de la estabilidad de los paquetes. Un paquete debe depender sólo de los paquetes que son más estables que él* [Martin, 1997b].

El ***principio de las abstracciones estables*** se enuncia como sigue: *los paquetes que son estables al máximo deben ser abstractos al máximo. Los paquetes inestables deben ser concretos. La abstracción de un paquete debe ser proporcional a su estabilidad* [Martin, 1997b].

Bibliografía

- ***Citada en las transparencias del tema:***

[Abbott, 1983] Abbott, R. J. "*Program Design by Informal English Descriptions*". Communications of the ACM, 26(11):882-894. November, 1983.

[Ancona et al., 1992] Ancona, D., Astesiano, E. and Zucca, E. "*Towards a Classification of Inheritance Relations*". Technical Report, Dipartimento di Informatica e Scienze dell'Informazione, Genova. 1992.

[Beck and Cunningham, 1989] Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Budd, 1991] Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Crespo, 2000] Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

[Firesmith et al., 1998] Firesmith, Donald, Henderson-Sellers, Brian and Graham, Ian. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

[Graham et al., 1997] Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

[Jacobson, 1987] Jacobson, Ivar. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Joyanes, 1998] Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Halbert and O'Brien, 1987] Halbert, D. and O'Brien, P. "*Using Types and Inheritance in Object-Oriented Programs*". IEEE Software, pages 71-79. 1987.

[Liskov, 1987] Liskov, Barbara. "*Data Abstraction and Hierarchy*". In Addedum to Proceedings of OOPSLA'87. Pages 17-35. ACM Press, 1987.

[Marqués, 1995] Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Oestereich, 1999] Oestereich, Bernd. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Sakkinen, 1989] Sakkinen, M. "*Disciplined Inheritance*". In Proceedings of ECOOP'89. Cook, S. editor. Pages 39-56. Cambridge University Press, 1989.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

[Wirfs-Brock et al., 1990] Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

- ***Lecturas complementarias:***

Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

Artículo que introduce las tarjetas de clase o tarjetas CRC.

Binder, Robert. "*Verifying Class Associations*". Object Magazine, 7(9):18-20. November, 1997.

Comenta los fallos más frecuentes en el diseño de asociaciones, así como comprobar su corrección.

Cardelli, Luca and Wegner, Peter. "*On Understanding Types, Data Abstraction and Polymorphism*". Computing Surveys, 17(4):471-523. December, 1985.

Artículo clásico de la teoría de tipos, donde se explica y se clasifica el polimorfismo dentro de los lenguajes de programación.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*El Principio Abierto/Cerrado*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(6):52-56. Junio, 1998.

Artículo en el que se explica el principio de diseño abierto/cerrado, apoyándose en un ejemplo en C++.

García Peñalvo, Francisco José y Marqués Corral, José Manuel. "*El Principio de Sustitución de Liskov*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(7):40-44. Julio, 1998.

Artículo que explica el principio de sustitución de Liskov, ilustrándolo con un ejemplo en C++.

García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre, 1997.

Informe técnico sobre las actividades a llevar a cabo en el desarrollo de elementos reutilizables bajo el paradigma objetual.

Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

Artículo crítico con las asociaciones bidireccionales tal y como se presentan en UML 1.x o en OMT. Se aboga por modelarlas mediante pares de asociciones unidireccionales (*mappings*).

Henderson-Sellers, B. "*OPEN Relationships- Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December, 1997.

Artículo que describe las relaciones de meronimia desde una perspectiva diferente a como lo hace UML.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Son muchos los capítulos de este libro cuya lectura puede complementar la exposición realizada en el tema. No obstante, se recomienda encarecidamente la lectura de su capítulo 24, **Utilizando bien la herencia**, donde se presenta una taxonomía de la herencia en doce categorías.

Parsons, Jeffrey and Wand, Yair. "*Choosing Classes in Conceptual Modeling*". Communications of the ACM, 40(6):63-69. June, 1997.

Artículo en el que se trata el problema de la identificación de las clases del dominio del problema.

Rumbaugh, James. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). ACM. Reprinted in ACM SIGPLAN 22(12):466-481. October, 1987.

Artículo en el que Rumbaugh presenta los problemas para la implementación de las relaciones, a excepción de la herencia, en los lenguajes orientados a objetos.

Rumbaugh, James. "*Depending on Collaborations: Dependencies as Contextual Associations*". Journal of Object-Oriented Programming (JOOP), 11(4):5-9. July/August, 1998.

Artículo que aboga por la distinción de las relaciones estructurales de las relaciones procedurales.

Rumbaugh, J., Jacobson, I. and Booch, G. "*The Unified Modeling Language Reference Manual*". Addison-Wesley, 1999.

Libro para consultar cualquier duda sobre el significado de los elementos que se utilizan en los diagramas UML.

Wegner, Peter. "*Dimensions of Object-Oriented Modeling*". IEEE Computer, 25(10):12-20. October, 1992.

Artículo que establece cuáles son los fundamentos de la programación orientada a objetos.

- ***Referencias utilizadas para preparar las clases:***

Al-Ahmad, W. and Steegmans, E. "*Inheritance in Object-Oriented Languages: Requirements and Supporting Mechanisms*". Journal of Object-Oriented Programming (JOOP), 12(8): 15-24,48. January, 2000.

Artículo donde se presentan dos tipos de herencia: *herencia de extensión* y *herencia de especialización*.

Arnold, Ken and Gosling, James. "*El Lenguaje de Programación Java*". Addison-Wesley/Domo, 1997.

Referencia de consulta sobre el modelo objeto que implementa Java. Es la traducción de [Arnold and Gosling, 1996].

Booch, Grady. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

En su tercer capítulo, **Clases y objetos**, se trata de una manera sobresaliente la implementación en C++ de las relaciones entre clases.

Budd, Timothy. "*Programación Orientada a Objetos*". Addison-Wesley Iberoamericana, 1994.

Libro muy adecuado para el desarrollo del tema porque en sus capítulos va desgranando los elementos del modelo objeto, estudiando como son soportados por diferentes lenguajes de programación orientados a objetos.

Coad, Peter. "*Finding Objects: Practical Approaches*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'91. (October 6 - 11, 1991, Phoenix, AZ USA). Pages 17-19. ACM, 1991.

Breve disquisición sobre las experiencias prácticas del autor en el descubrimiento de los objetos.

Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

Su segundo capítulo, **Clases, tipos, herencia y subtipado**, es un sensacional tratado y estado del arte de la teoría de tipos en su relación con la Orientación a Objetos. Las relaciones tipo/clase y subtipo/herencia se presentan de una manera sobresaliente.

Champeaux, Dennis de, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison-Wesley, 1993. HTML Edition available at http://gee.cd.oswego.edu/dl/oosdw3. [Última vez visitado, 1-2-2000].

Adecuada referencia para el estudio de las relaciones en el modelo objeto, tanto en la fase de análisis como en la de diseño.

Deadman, Richard. "*When in Rome: A Guide to the Java Paradigm*". Java Report, 2(9):41-52,70. October, 1997.

Artículo que resume las características orientadas a objetos de Java que se deben conocer cuando se migra a este lenguaje desde C++.

García Peñalvo, Francisco José y Crespo González-Carvajal, Yania. "*Las Implicaciones de Eiffel en el Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(7):45-53. Julio, 1998.

Artículo que presenta las características de Eiffel como un lenguaje de programación orientada a objetos diseñado que conjuga el paradigma objetual con los principios de la Ingeniería del Software, convirtiéndose en un marco de trabajo adecuado para expresar el diseño de los sistemas software.

Graham, Ian. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

En su capítulo 1, **Conceptos básicos**, se presentan las bases de un modelo objeto.

Johnson, Ralph E. and Foote, Brian. "*Designing Reusable Classes*". Journal of Object-Oriented Programming, 1(2):22-35. 1988.

Artículo sobre las técnicas a emplear para la creación de clases reutilizables.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

La mayor parte de los capítulos de este libro se ajustan a los contenidos del presente tema. Realiza un análisis crítico de las características de tres lenguajes orientados a objetos fundamentales: *Java, Eiffel* y *C++*.

Katrib Mora, Miguel. "*C++ Versus Eiffel (I)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(10):73-78. Noviembre, 1996.

Compara las clases y los mecanismos de creación de objetos en Eiffel y C++.

Katrib Mora, Miguel. "*C++ Versus Eiffel (II)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(11):69-76. Diciembre, 1996.

Compara la herencia y el polimorfismo en estos lenguajes.

Katrib Mora, Miguel. "*C++ Versus Eiffel (y III)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, IV(1):67-74. Enero, 1997.

Aborda los problemas de la herencia repetida y de la genericidad comparando Eiffel y C++.

Larman, Craig. "*Designing Object Responsibilities and Collaborations*". Java Report, 5(3):69-75. March, 2000.

Modelado de responsabilidades y colaboraciones en UML, en lugar de utilizar las tarjetas CRC.

Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

Obra fundamental para entender los problemas de la herencia múltiple.

Martin, Robert C. "*The Open Closed Principle*". C++ Report, 8(1). January, 1996.

Introducción al principio abierto cerrado.

Martin, Robert C. "*The Liskov Substitution Principle*". C++ Report, 8(3). March, 1996.

Presentación del principio de sustitución del Liskov.

Martin, Robert C. "*The Dependency Inversion Principle*". C++ Report, 8(5). May, 1996.

Presentación del principio de inversión de dependencias.

Martin, Robert C. "*The Interface Segregation Principle*". C++ Report, 8(7). July-August, 1996.

Presentación del principio de separación de la interfaz.

Martin, Robert C. "*Granularity*". C++ Report, 8(10). November-December, 1996.

Presentación de los siguientes principios: equivalencia reutilización/revisión, cierre común, reutilización común y dependencia acíclica.

Martin, Robert C. "*Stability*". C++ Report, 9(2). February, 1997.

Presentación del principio de las dependencias estables y del principio de las abstracciones estables.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Como se indicó en las lecturas complementarias, varios capítulos de este libro se pueden utilizar para la impartición de este tema, en concreto se destacan los siguientes capítulo: **6 Tipos abstractos de datos; 7 La estructura estática: las clases; La estructura de ejecución: los objetos; 14 Introducción a la herencia; 15 Herencia múltiple.**

Oaks, Scott. "*Creating Good Java Interfaces*". Java Report, 4(12):88,86. December, 1999.

Sobre las interfaces java.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Guía de referencia a UML 1.3. Podría utilizarse cualquiera de las referencias citadas sobre UML en el tema 7 de la asignatura de Ingeniería del Software.

Rodríguez, Juan José, Crespo, Yania y Marqués, Corral. "*Transformación de Jerarquías de Herencia Múltiple en Jerarquías de Herencia Sencilla*". Technical Report TR-GIRO-03-98. Universidad de Valladolid. 1998.

Trabajo en el que se analizan situaciones en las que se plantea la necesidad de transformar esquemas que utilizan jerarquías de herencia múltiple en esquemas equivalentes que utilicen herencia sencilla.

Rumbaugh, James. "*Controlling Propagation of Operations Using Attributes on Relations*". In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications – OOPSLA'88. (September 25 - 30, 1988, San Diego, CA USA). Pages 285-296. ACM, 1988.

Estrategias para controlar la propagación de operaciones a través de una colección de objetos conectados por varias asociaciones.

Rumbaugh, James, Blaha, **Michael, Premerlani, William, Eddy, Frederick, Lorensen, William**. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Libro imprescindible para la comprensión del modelo objeto. Con relación al tema tratado se van a destacar los siguientes capítulos: **3 Modelado de objetos**, donde se introducen los elementos del modelo objeto; **10 Diseño de objetos**, en el que se va exponiendo el refinamiento que sufren los objetos semánticos para adecuarse a la solución, explicando diferentes alternativas para su futura implementación; **15 Lenguajes orientados a objetos**, donde, con las construcciones propias de algunos LPOO, se presenta cómo implementar los elementos del modelo objeto.

Stroustrup, Bjarne. "*El Lenguaje de Programación C++*". 3ª Edición. Addison-Wesley, 1998.

Dada la importancia que se ha dado al lenguaje C++ en la asignatura, este libro no podía faltar como referencia básica a las construcciones del lenguaje. Es la traducción de [Stroustrup, 1997].

Tema 4: *Genericidad*

Descriptores

Genericidad; tipo paramétrico; clase genérica (paramétrica); programación genérica; plantilla, STL, diseño para reutilización.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T7, T9** y **P2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.
- Estudio y comprensión de los fundamentos del diseño de sistemas software.
- Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentación de la parametrización de tipos como una dimensión horizontal para aumentar la extensión, reutilización y fiabilidad de las clases.
- Introducción las nociones básicas de la programación genérica.
- Estudio de las construcciones sintácticas de C++ relacionadas con la genericidad.
- Introducción a la biblioteca estándar de plantillas de C++ (STL – Standard Template Library).

Contenidos

4.1 Introducción
4.2 Tipos paramétricos y clases genéricas
4.3 Programación paramétrica
4.4 Genericidad en C++

Tabla 5.28. Contenidos del cuarto tema del programa teórico de Programación Orientada a Objetos

Resumen

La genericidad en la Orientación a Objetos se aborda de una forma introductoria en este cuarto tema, para lo que se han considerado cuatro apartados que se introducen someramente a continuación.

El primer apartado sirve para introducir la noción de genericidad, ya que es un concepto nuevo, del que sólo se ha hecho alguna breve reseña en la asignatura de Ingeniería del Software y en los temas anteriores de esta asignatura.

Para alcanzar las metas de extensibilidad, reutilización y fiabilidad es preciso hacer que la estructura de clase sea más flexible, y este esfuerzo está dirigido en dos direcciones. Una, vertical que representa abstracción y especialización; para lo que se

utiliza la herencia. Otra, horizontal, que viene de la mano de la parametrización de tipos, conocida también como genericidad, como se muestra en la Figura 5.10 [Meyer, 1997].

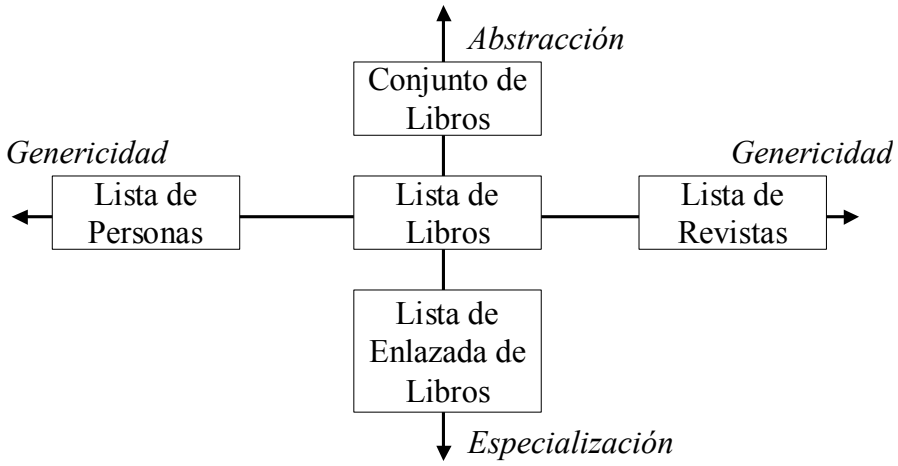

Figura 5.10. Dimensiones de la generalización

La genericidad es la posibilidad de definir módulos parametrizados, y puede hallarse en muchos lenguajes que soportan el encapsulamiento, como Ada. Usualmente, los parámetros son tipos. Los mensajes genéricos permiten la creación de componentes reutilizables y de marcos de trabajo, eliminando la dependencia de la rutina invocada con respecto a aquello que se invoca.

Las clases, como noción central de la tecnología de objetos, donde una clase se ve como un producto mezcla entre el concepto de módulo y el concepto de tipo. Mientras no se tenía genericidad se podía decir que toda clase es un módulo y también tipo.

Con la genericidad, la segunda de las afirmaciones no es literalmente cierta, aunque la diferencia sea pequeña. Una clase genérica declarada como *C[G]* es, en lugar de un tipo, un patrón de tipo que abarca un conjunto infinito de tipos posibles; se puede obtener uno de estos tipos al proporcionar un parámetro genérico real, que es a su vez un tipo, que corresponde a *G*.

El segundo apartado presenta las definiciones básicas de los conceptos relacionados con la genericidad, justificando la necesidad de la parametrización de tipos.

En la teoría de tipos abstractos de datos la parametrización aparece como algo natural. Así, al definir un TAD pila, no se definen pilas de enteros, sino pilas genéricas, *pila[G]*, donde *G* es conocido como un parámetro formal e indica cualquier tipo contenible en la pila. De una vez se están definiendo todos los tipos de pila existentes (*en función del tipo de sus componentes*).

Se denomina tipo paramétrico o genérico *a un tipo cuya especificación está definida en función de alguna variable de tipo, libre para ser instanciada* [Crespo, 2000].

Un tipo paramétrico es, por tanto, un tipo que describe tipos. Ya que los tipos a su vez describen objetos, un tipo paramétrico es también una descripción de objetos. Un tipo paramétrico describe tipos que tienen en común estructura y comportamiento pero que se diferencian en el tipo de algunas de sus propiedades. La especificación de una descripción común a todos se logra a través de las variables de tipos. Una sustitución para las variables de tipo de un tipo paramétrico da lugar a otro tipo, instancia del tipo paramétrico.

Las clases genéricas son clases que se definen en función de uno o más parámetros de tipo, los parámetros genéricos formales. Estos parámetros se utilizan en las signaturas y en el cuerpo de la clase genérica como anotaciones de tipo. Las clases genéricas implementan tipos paramétricos en un lenguaje de programación orientado a objetos.

Así, una clase genérica puede definirse como *la construcción lingüística en los lenguajes orientados a objetos que se corresponde con la implementación de un tipo paramétrico* [Crespo, 2000].

Para las clases genéricas se utilizan distintos nombres en distintos lenguajes de programación. Como clases genéricas se conocen en Eiffel, como plantillas (*templates*) en C++, como módulos genéricos en Modula-3 [Cardelli et al., 1989] entre otros.

No todos los lenguajes de programación orientados a objetos incluyen la posibilidad de definir clases genéricas en las construcciones del lenguaje. Lenguajes como Smalltlak son un claro ejemplo, dado que no es un lenguaje estáticamente tipado. Las clases genéricas sólo tienen sentido en lenguajes estáticamente tipados. Java es un lenguaje que sí es estáticamente tipado, pero que tampoco incluye clases genéricas. Otra familia que cabe ser mencionado es la de los lenguajes funcionales orientados a objetos estáticamente tipados, como Haskel [Thompson, 1999] por ejemplo; donde aunque no esté presente una construcción lingüística especial para definir clases genéricas, la implementación de algunos tipos paramétricos se obtiene implícitamente. Esto se logra con la definición de clases donde hay ausencia de anotaciones de tipo. En estos casos, es el mecanismo de inferencia de tipos el que detecta si una entidad puede denotar objetos de cualquier tipo.

El tercer apartado introduce el concepto de programación paramétrica o programación genérica. Mediante esta técnica, se busca la definición de algoritmos y estructuras de datos en un nivel abstracto o genérico [Musser and Stepanov, 1989a].

Con esta técnica los módulos pueden tener parámetros definidos sobre una interfaz bien general que además describa exactamente qué propiedades se requieren en un entorno para que el módulo funcione correctamente. La reusabilidad se potencia por el mecanismo que permite que otros módulos instancien los parámetros.

La idea básica detrás de esta técnica es maximizar la reusabilidad de los módulos, almacenándolos en la forma más general posible. Los módulos tienen parámetros y

entonces se pueden construir nuevos módulos a partir de los existentes, instanciando los parámetros. Como el propio término lo indica, los bloques constructivos básicos de la programación paramétrica son los módulos paramétricos [Goguen, 1984]. Dependiendo del lenguaje, el término módulo puede ser cambiado por tipo, función, clase, paquete... y los requisitos de los parámetros por acotaciones.

Con origen en el trabajo de **J. Goguen** [Goguen, 1984], posteriores trabajos han dado lugar a una disciplina que se ha dado en llamar Programación Genérica o Paramétrica [Musser and Stepanov, 1989a]. **David Musser y Alexander Stepanov** definen cuatro tipos de abstracciones en las que se trabaja en esta disciplina, abstracciones de datos, de algoritmos, de estructura y de representación.

Recientemente en [Musser and Stepanov, 1998] se da una definición precisa de la disciplina de la Programación Genérica, definiéndose como *un área que tiene que ver con tratar de encontrar representaciones genéricas de algoritmos eficientes, estructuras de datos y otros conceptos de software y con su organización sistemática. El objetivo es expresar algoritmos junto con las estructuras de datos en una forma ampliamente adaptable e interoperable que permita su uso directo en la construcción de software. Las ideas claves incluyen expresar algoritmos concretos a un nivel lo más general posible sin perder eficiencia, es decir, que cuando la forma general se instancie para obtener los casos concretos resulten algoritmos tan eficientes como los originales*.

La programación genérica depende de que los lenguajes permitan hacer descomposición de programas en componentes genéricos que puedan ser desarrollados de forma separada y combinados arbitrariamente, sujetos a interfaces bien definidas. La base de la interrelación entre abstracciones genéricas de datos y abstracciones de algoritmos en la programación genérica son los iteradores.

Según [Ghezzi and Jazayeri, 1998], la aproximación a la programación genérica utilizando iteradores fue originalmente promovida por la biblioteca STL (Standard Template Library) [Stepanov and Lee, 1994], [Musser and Saini, 1996], que actualmente forma parte de la biblioteca estándar de C++. Sin embargo, este planteamiento es injusto con las ideas promovidas por **Liskov** y sus colaboradores en el diseño e implementación del lenguaje CLU y su biblioteca predefinida; ya que en CLU se definen iteradores como un recurso del lenguaje y se vinculan con los tipos paramétricos para definir abstracciones e implementaciones altamente reutilizables [Liskov, 1993].

En el cuarto y último apartado se aborda la genericidad dentro de un lenguaje concreto. Para ello se introducen por separado las plantillas (*templates*), como construcciones propias del lenguaje para la definición de módulos genéricos (funciones

y clases) [Stroustrup, 1997], y por otro lado la biblioteca estándar de plantillas, STL[28], integrada ya en el estándar ANSI de C++ [X3J16/WG21, 1996].

Las plantillas de C++ proporcionan una forma sencilla de representar una amplia gama de conceptos generales y una forma sencilla de combinarlos. Las clases y funciones resultantes pueden equipararse a código más especializado escrito a mano en cuanto a tiempo de ejecución y eficiencia de espacio.

Las plantillas proporcionan soporte directo para la programación genérica. El mecanismo de plantillas de C++ permite que un tipo sea un parámetro en la definición de una clase o de una función. Una plantilla depende sólo de las propiedades que usa realmente de sus tipos de parámetros, no requiriendo que los tipos diferentes usados como argumentos estén relacionados explícitamente.

La biblioteca estándar de plantillas (STL) es una biblioteca de clases genéricas para C++ desarrollada por **Alexander Stepanov** y **Meng Lee** en los laboratorios de Hewlett Packard en Palo Alto, California [Stepanov and Lee, 1994], [Stepanov and Lee, 1995].

Los orígenes de esta biblioteca de clases están en los trabajos que sobre programación genérica estaban llevando a cabo **Alexander A. Stepanov** y **David R. Musser** desde finales de la década de setenta. En 1985 estos dos investigadores desarrollan una biblioteca de funciones genéricas para Ada [Musser and Stepanov, 1987], [Musser and Stepanov, 1989b]. Por aquel entonces este proyecto no era viable en C++ porque este lenguaje no contaba con las construcciones lingüísticas adecuadas para soportar genericidad, las plantillas. En 1988 **Stepanov** se incorpora a Hewlett Packard, y en 1992 dirige un proyecto sobre algoritmos genéricos. Dentro de este proyecto, **Alexander Stepanov** y **Meng Lee** desarrollan la biblioteca STL con la intención de demostrar que se pueden tener algoritmos definidos de forma tan genérica como sea posible sin pérdida de eficiencia. En 1993 **Musser** y **Stepanov** desarrollan un informe técnico sobre los principios de diseño para la construcción y documentación de una biblioteca basada en algoritmos genéricos; no describe STL en sí, pero presenta ejemplos de una biblioteca que podría considerarse una antecesora de la propia STL, este artículo aparece publicado un año más tarde [Musser and Stepanov, 1994].

La biblioteca STL fue puesta como dominio público, siendo incluida en el borrador del estándar de ANSI/ISO C++ el 14 de julio de 1994.

El código C++ de esta biblioteca es de una gran calidad, no haciendo uso de la herencia en ningún momento.

STL es una biblioteca de componentes. Estos componentes pueden ser de cinco tipos:

[28] La biblioteca STL se introduce someramente en este tema, y será objeto de un estudio más detallado en el temario de prácticas.

- ***Algoritmo:*** Procedimientos computacionales que pueden trabajar sobre diferentes contenedores.
- ***Contenedor:*** Objeto que puede contener y manejar objetos.
- ***Iterador:*** Abstracción de un algoritmo de acceso a contenedores, por tanto un algoritmo que puede trabajar sobre diferentes contenedores.
- ***Objeto función:*** Clase que tiene definida la función *operator()*.
- ***Adaptador:*** Encapsula un componente para ofrecer otra interfaz.

Bibliografía

- ***Citada en las transparencias del tema:***

[Crespo, 2000] Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Stepanov and Lee, 1994] Stepanov, A. A. and Lee, M. "*The Standard Template Library*". Technical Report HPL-94-34, Hewlett-Packard Laboratories, April 1994.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

- ***Lecturas complementarias:***

Devis Botella, Ricardo. "*Contenedores y Plantillas en C++*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(5):61-71. Marzo, 1995.

Artículo sobre las facilidades ofrecidas por C++ para la creación de contenedores parametrizables empleando plantillas.

Kirman, Jak. "*A Modest STL Tutorial*". http://www.cs.brown.edu/people/jak/proglan/stltut/tut.html. [Última vez visitado, 10-11-1999]. January 1998.

Sencillo tutorial sobre STL. Adecuado para introducirse y hacer los primeros ejemplos.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

El capítulo 10, **Genericidad**, contiene una introducción a la genericidad.

Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". In Proceedings of the First International Joint Conference of ISSAC-88 and AAECC-6. P. Gianni editor. (Rome, Italy, July 4-8, 1988). Published in *Lecture Notes in Computer Science* 358, Pages 13-25. Springer-Verlag, 1989.

Introducción a la programación genérica. Artículo centrado en las abstracciones algorítmicas más que en las estructuras de datos.

Musser, David R. and Stepanov, Alexander A. "*Algorithm-Oriented Generic Library*". Software Practice & Experience, 24(7):623-642. July, 1994.

Trabajo sobre bibliotecas de algoritmos genéricos, predecesor a STL.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". STL Documentation. October. 1995.

Manual que se incluía con la distribución pública de la biblioteca STL.

- ***Referencias utilizadas para preparar las clases:***

Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

Su tercer capítulo, **Tipos paramétricos y clases genéricas**, trata de una manera sobresaliente los problemas de los tipos paramétricos, haciendo una corta, pero brillante introducción a la programación genérica.

Crespo, Yania, Marqués, Corral y Rodríguez, Juan José. "*Genericidad Inversa*". En las Actas de las II Jornadas de Trabajo MENHIR. Editor J. Á. Carsí. (19-20 de febrero de 1998. Valencia - España). Páginas 143-148. Organizado por el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. Febrero, 1998.

Propuesta de un operador para convertir clases en clases genéricas de forma automática, potenciando de esta manera la reutilización del software.

Crespo, Yania, Marqués, Corral y Rodríguez, Juan José. "*Transformación de Clases para Obtener Clases Genéricas. Implicaciones en las Jerarquías de Herencia y de Agregación/Composición*". Technical Report TR-GIRO-04-98. Universidad de Valladolid. 1998.

Informe técnico que presenta aspectos avanzados de la genericidad dentro de un modelo objeto.

Crespo, Yania, Rodríguez, Juan José, García, Francisco José y Marqués, José Manuel. "*Obtención Automática de Clases Genéricas a través de una Operación de Parametrización*". En Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISDB'99. Editores Pere Botella, Juan Hernández y Félix Saltor. (Cáceres, 24-26 de Noviembre de 1999):343-354. 1999.

Se presenta la definición de un operador de reestructuración de clases que permite obtener clases genéricas a partir de clases que no lo son. A este operador se le ha denominado operador de parametrización.

Devis Botella, Ricardo. "*C++. STL, Plantillas, Excepciones, Roles y Objetos*". Paraninfo, 1997.

Libro que introduce, pero no profundiza, en temas tan interesantes como la genericidad, las excepciones o los roles en C++.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

En su sexto capítulo, **Type Extension: Generics and Templates**, compara la genericidad en C++ y en Eiffel.

Katrib Mora, Miguel. "*C++ Versus Eiffel (y III)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, IV(1):67-74. Enero, 1997.

Aborda los problemas de la herencia repetida y de la genericidad comparando Eiffel y C++.

Prieto Arambillet, Félix. "*Apuntes de la Asignatura Programación III*". Versión 1.0. Departamento de Informática. Universidad de Valladolid. Diciembre, 1999.

Uno de los temas está dedicado a la genericidad en Eiffel.

Reeves, Jack W. "*More on Template Specialization*". C++ Report, 12(2):16-20. February, 2000.

Artículo que defiende el uso de las plantillas en los desarrollos con C++, aclarando ciertos mitos o malentendidos.

Rensselaer Polytechnic Institute. "*Standard Template Library Reference*". Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/~musser/stl. [Última vez visitado, 6-3-2000]. 1994.

Completa referencia a la biblioteca STL.

Smith, Mark L. "*An STL-Based N-Way Set*". C/C++ Users Journal, 18(3):76-83. March, 2000.

Ejemplo avanzado del uso de la biblioteca de plantillas estándar de C++ (STL).

Stroustrup, Bjarne. "*El Lenguaje de Programación C++*". 3ª Edición. Addison-Wesley, 1998.

Su capítulo 13, **Plantillas**, está dedicado al mecanismo lingüístico de C++ relacionado con la genericidad. La tercera parte del libro, como su propio nombre indica, **La biblioteca estándar** (*capítulos 16 a 22*), está centrada en el estudio de la misma, incluyendo por tanto a la biblioteca STL.

Sutter, Herb. "*Standard Library News, Part 1: Vectors and Deques*". C++ Report, 11(7). July/August, 1999.

Sutter, Herb. "*Standard Library News: Sets and Maps*". C++ Report, 11(9):38-41. October, 1999.

En estos dos artículos el autor explora algunas características de los contenedores en la biblioteca STL.

Weidl, Johannes. "*The Standard Template Library Tutorial*". Technical Report 184.437 Wahlfachpraktikum (10.0). Information Systems Institute. Distributed Systems Department. Technical University Vienna. http://www.infosys.tuwien.ac.at/Research/Component/Tuturial/prwmain.htm. [Última vez visitado, 6-3-2000]. April, 1996.

Tutorial introductorio a la biblioteca STL. Muy completo.

Tema 5: ***Manejo de Excepciones***

Descriptores

Excepción; software robusto; fallo; clase de excepción.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T7, T9** y **P2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.
- Estudio y comprensión de los fundamentos del diseño de sistemas software.
- Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Aprender a manejar sucesos inesperados en tiempo de ejecución.
- Diseño de una política de tratamiento de excepciones adecuada.
- Conocer los mecanismos para el manejo de excepciones de C++.

Contenidos

5.1 Conceptos básicos
5.2 Tratamiento de excepciones
5.3 Mecanismo de excepciones
5.4 Guías para el diseño de excepciones

Tabla 5.29. Contenidos del tema 5 del programa teórico de Programación Orientada a Objetos

Resumen

Por muchas precauciones estáticas que se tomen en la construcción de un sistema software, pueden suceder acontecimientos inesperados y no deseados durante su ejecución, los cuales reciben el nombre de excepciones, y en pro de un sistema robusto se debe estar preparado para tratarlas.

El tratamiento de las excepciones es el centro del presente tema, que se ha dividido en cuatro apartados principales que se resumen a continuación.

En el primer apartado, **conceptos básicos**, se introduce el problema del tratamiento de las excepciones. Un primer problema es el mal uso de los mecanismos de excepción que hacen algunos lenguajes de programación, como PL/1 o Ada, donde en lugar de reservarse para casos anormales, sirven a fin de cuentas como instrucciones de salto incondicional entre rutinas, lo que viola fragantemente el principio de protección modular.

Informalmente una excepción es un suceso anormal que rompe la ejecución de un sistema. Se dice que una llamada a una rutina tiene éxito si termina su ejecución en un

estado en el que satisface el *contrato* de la rutina. Fracasa si no tiene éxito. Un *fracaso* de una rutina causa una *excepción* en quien la llama. Una llamada a una rutina fracasa si y sólo si ocurre una excepción durante la ejecución de ésta y la rutina no se puede recuperar de dicha excepción.

Las excepciones proporcionan una manera limpia de verificar errores sin abarrotar el código. Las excepciones proporcionan además un mecanismo para señalar directamente los errores, en lugar de usar indicadores o efectos colaterales tales como campos que deben ser verificados. Las excepciones hacen que las situaciones de error que puede señalar un método sean parte explícita del contrato del mismo. La lista de excepciones puede ser vista por el programador, verificada por el compilador y conservada por las clases extendidas que anulan el método.

Se lanza una excepción cuando e encuentra una situación imprevista de error. La excepción es capturada entonces por una cláusula situada más arriba en la pila de invocación al método.

Una discusión abierta en la comunidad de la tecnología de objetos es si las excepciones deben ser objetos. En lenguajes como Java o Delphi las excepciones son objetos, mientras que en otros lenguajes, como Eiffel, esto no sucede así, y las excepciones no son ciudadanos de primera clase.

En el segundo apartado se aborda el tratamiento de las excepciones; para ello, y siguiendo el ejemplo de [Meyer, 1997], se comienza el apartado con unos contraejemplos que ilustren el mal uso de las excepciones. Tras lo cual ya se está en condición de enunciar el principio disciplinado de tratamiento de excepciones [Meyer, 1997], que dice: *sólo hay dos respuestas legítimas a una excepción que ocurra durante la ejecución de una rutina:*

R1 – Reintento.- *intentar cambiar las condiciones que condujeron a la excepción y ejecutar de nuevo la rutina desde el comienzo.*

R2 – Fracaso.- *limpiar el entorno, terminar la llamada e informar del fallo a quien hiciera la llamada.*

El tercer apartado repasa el mecanismo de excepciones de C++ mediante las construcciones *try, catch* y *throw*. Un mecanismo muy parecido a C++ es el de Java. Por su parte Eiffel presenta un mecanismo mejor integrado con el lenguaje, y que es fundamental en el diseño por contrato.

Por último, en el cuarto apartado se dan una serie de reglas para diseñar un buen mecanismo de tratamiento de excepciones.

Bibliografía

- ***Citada en las transparencias del tema:***

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

- ***Lecturas complementarias:***

Devis Botella, Ricardo. "*Manejo de Excepciones en C++*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(2):45-53. Febrero, 1996.

Artículo sobre el tratamiento de excepciones en C++.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

El capítulo 12, **Cuando se rompe el contrato: Tratamiento de excepciones**, aborda el manejo de los mecanismos de excepción sobre la base de Eiffel, esto es, sobre la base del diseño por contrato.

Rojas, Alfonso y Lozano, Javier. "*Excepciones en C++ para DOS. La Última Herramienta para el ANSI C++*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(6):43-47. Abril, 1995.

Artículo introductorio a la excepciones en C++.

- ***Referencias utilizadas para preparar las clases:***

Arnold, Ken and Gosling, James. "*El Lenguaje de Programación Java*". Addison-Wesley/Domo, 1997.

El capítulo 7, **Excepciones**, es una presentación del tratamiento de excepciones en Java.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Dentro de su capítulo 11, **Run time**, dedica un apartado para repasar someramente los mecanismos de excepciones de C++, Java y Eiffel.

Stroustrup, Bjarne. "*El Lenguaje de Programación C++*". 3ª Edición. Addison-Wesley, 1998.

Su capítulo 14, **Manejo de excepciones**, es una referencia completa sobre el manejo de excepciones en C++.

Unidad Docente III: Diseño Orientado a Objetos Avanzado

Objetivo genérico

Esta unidad docente tiene el objetivo tratar aspectos más profundos y de mayor complejidad, haciendo uso de los fundamentos de diseño presentados en la unidad docente dos.

Son muchos los aspectos del diseño orientado a objetos que podrían tener cabida en una unidad docente como esta, pero contando siempre con los límites temporales en los que se desarrolla la asignatura y el perfil de los discentes, alumnos del tercer curso de una Ingeniería Técnica en Informática de Sistemas que se están introduciendo al DOO con esta asignatura, se han elegido dos temas: *los patrones de diseño* y *el diseño por contrato*; en detrimento de otros temas, también interesantes, como podían ser el diseño de *frameworks*, la relación entre la genericidad, la herencia y el polimorfismo, los objetos distribuidos, los modelos de componentes, las métricas en la Orientación a Objetos, el modelado de roles, la migración de objetos, el diseño de la concurrencia o la persistencia.

Los patrones de diseño se han elegido por ser una de las formas más elegantes de reutilizar la experiencia de expertos que se han enfrentado a problemas que aparecen recurrentemente en el diseño de los sistemas software.

En los últimos cuatro años se ha justificado en diversas conferencias internacionales la presencia de los patrones software en las líneas de investigación y trabajo dentro de la Programación Orientada a Objetos [Guerraoui et al., 1996], así como en el currículo de un ingeniero informático [Astrachan et al., 1998], utilizándolos en diversas disciplinas como pueden ser análisis y diseño orientado a objetos [Cybulski, and Linden, 2000], programación [Wallingford, 1996], [Parrish et al., 1997], [Fell et al., 1998], estructuras de datos [Gelfand et al., 1998], [Nguyen, 1998] o incluso como herramienta pedagógica [Clancy and Linn, 1999].

El diseño por contrato es una de las bases más importantes de Eiffel, ofreciendo una forma segura de construir sistemas software.

El objetivo es introducir ambos temas, de forma que el alumno, si lo cree necesario tenga un camino abierto para profundizar en ellos.

Como sucedía con la unidad docente dos, la presente unidad docente está muy relacionada con la parte práctica de la asignatura, donde C++ es el lenguaje de programación orientado a objetos que se está utilizando para llevar a la práctica los conceptos tratados en la parte teórica.

Esta unidad docente supone el **37%** del programa teórico de la asignatura Programación Orientada Objetos, compuesta por dos temas: **Patrones de diseño**; y **Diseño por contrato** que se detallan a continuación.

Tema 6: *Patrones de Diseño*

Descriptores

Patrón software (*pattern*); patrón de diseño (*design pattern*); antipatrón (*antipattern*); patrón arquitectónico; patrón de creación; patrón estructural; patrón de comportamiento; *idiom*; diseño orientado a objetos; reutilización; elemento reutilizable.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T7, T9** y **P2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.
- Estudio y comprensión de los fundamentos del diseño de sistemas software.
- Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducción del concepto de patrón software.
- Clarificar el origen de los patrones software.
- Uso de los patrones de diseño en la construcción de sistemas software de calidad.
- Relacionar los patrones software con la reutilización del software.
- Estudio de algún representante de los principales tipos de patrones de diseño.

Contenidos

6.1 Patrones software
6.2 Patrones arquitectónicos
6.3 Patrones de creación
6.4 Patrones estructurales
6.5 Patrones de comportamiento
6.6 Antipatrones

Tabla 5.30. Contenidos del tema 6 del programa teórico de Programación Orientada a Objetos

Resumen

Los patrones (*patterns*) pueden considerarse como unidades de información destinadas a soportar, documentar y transmitir la experiencia en la resolución de problemas tipo que pueden aparecer en diferentes contextos. Cuando el contexto de los problemas a estudiar es el desarrollo de software se está dentro de los denominados patrones software. El objetivo que se persigue con los patrones dentro del mundo del desarrollo del software es establecer un catálogo de referencia para ayudar a los ingenieros de

software a solucionar o a afrontar problemas de Ingeniería del Software, dando lugar a un lenguaje común en el que comunicar la experiencia y los trucos entorno a dichos problemas y a su solución [García, 1998].

En el presente tema se estudian los patrones software, más concretamente los patrones de diseño, para lo que el tema se ha organizado en seis apartados, como se aprecia en la Tabla 5.30.

El primer apartado sirve como introducción al tema de los patrones, como elementos reutilizables de experiencia y conocimiento que han calado profundamente en el área del desarrollo de aplicaciones software, teniendo su caldo de cultivo más activo en la comunidad de la tecnología de objetos. De este hecho se deriva el término patrón software y más concretamente el de patrón de diseño para hacer referencia al uso de patrones en el Diseño Orientado a Objetos.

El concepto de patrón software se difunde gracias a **Erich Gamma, Richard Helm, Ralph Jonhson** y **John Vlissides**, denominados popularmente como la Banda de los Cuatro (*The Gang of Four, o simplemente por GoF*) con su artículo [Gamma et al., 1993] y especialmente con la publicación del libro *Design Patterns: Elements of Reusable Object-Oriented Software* [Gamma et al., 1995].

Realmente el uso del término patrón, con el significado que actualmente se le da en la Ingeniería del Software, y más concretamente en el área de la tecnología de objetos, se deriva de los trabajos del arquitecto **Christopher Alexander** desde mediados de los años sesenta. El trabajo de Alexander se resume en varios libros y artículos sobre planificación de urbanismo y arquitectura de edificios [Alexander, 1964], [Alexander et al., 1975], [Alexander et al., 1977], [Alexander, 1979]. Sin embargo, aún siendo sus publicaciones propias de arquitectura, sus ideas son aplicables a muchas otras disciplinas, entre las que cabe destacar su aplicación en el desarrollo de software.

Las ideas de Alexander fueron utilizadas por **Ward Cunningham** y **Kent Beck** para desarrollar un lenguaje de patrones (compuesto por cinco patrones) como guía de iniciación a la programación en Smalltalk, presentando su trabajo en la OOPSLA'87 [Beck and Cunningham, 1988].

Posteriormente, **James Coplien** fue recopilando y creando un catálogo de *idioms* propios de C++ que fue posteriormente publicado en 1992 en su libro *Advanced C++ Programming Styles and Idioms* [Coplien, 1992].

Durante los años comprendidos entre 1990 y 1994 diversos *workshops* sobre el tema tuvieron lugar, especialmente en las diversas ediciones de la OOPSLA, hasta que en Abril de 1994 se decidió crear una conferencia internacional especializada en patterns: el **PloP** (*Pattern Languages of Program Design*), que desde 1994 tiene lugar en USA, desde 1996 en Europa, **EuroPLoP**, y desde este año en el Pacífico Sur, **KoalaPLoP**.

Durante este tiempo la *Banda de los Cuatro* había recopilado su trabajo, que sería publicado en el afamado libro del GoF [Gamma et al., 1995], siendo presentado en la OOPSLA'94. Este libro fue considerado como el mejor libro de Orientación a Objetos de 1995 y el mejor libro de orientación al objeto de todos los tiempos por la revista JOOP (*Journal of Object Oriented Programming*) en su número de septiembre de 1995; en 1998 la revista DrDobbs le otorga el premio de excelencia en programación.

No existe una definición estándar para el término patrón, así se pueden encontrar varias en la bibliografía. De hecho, parafraseando a **James Coplien**: "*Alexander podría haber escrito una frase de definición de lo qué es un patrón, pero en su lugar él escribió un libro de 550 páginas para hacerlo porque el concepto es difícil*".

Así, para **Trigve Reenskaug**, padre del método OOram, un patrón es *una descripción en un formato fijo de cómo solucionar un cierto tipo de problemas* [Reenskaug et al., 1996]. Para **Brad Appleton** un patrón es *una unidad de información instructiva con nombre que captura la estructura esencial y la comprensión de una familia de soluciones exitosas probadas para un problema recurrente que ocurre dentro de un cierto contexto y de un sistema de fuerzas* [Appleton, 2000]. Sin embargo, para **James Coplien** cada patrón es una *regla constituida por tres partes, la cual expresa una relación entre un cierto contexto, un cierto sistema de fuerzas que ocurren repetidamente en ese contexto, y una cierta configuración software que permite a estas fuerzas resolverse así mismas* [Coplien, 1998].

La definición de **James Coplien** sirve para introducir una serie de términos que configuran la terminología propia de los patrones. En primer lugar el término **sistema de fuerzas**, significa que un problema se refiere a un conjunto de fuerzas. La noción de **fuerza** generaliza el tipo de criterios que los ingenieros del software utilizan para justificar los diseños y las implementaciones. En el caso de los patrones, éstos casan con conjuntos de objetivos y restricciones que se encuentran en el desarrollo de cada elemento que se vaya a crear. Estas fuerzas son grandes, difíciles de medir y conflictivas. También es interesante resaltar el término **configuración software** en el sentido de un diseño en una forma canónica o un conjunto de reglas de diseño que alguien puede aplicar para solucionar las fuerzas del problema.

Como se desprende de las definiciones anteriores, un patrón involucra una descripción general de una solución recurrente para un problema recurrente repleto de diferentes objetivos y restricciones. Para **James Coplien** [Coplien, 1998] un buen patrón debe cumplir los siguientes requisitos:

- **Debe solucionar un problema**: Los patrones capturan soluciones, no sólo principios abstractos o estrategias.
- **Son conceptos probados**: Los patrones capturan soluciones que han sido probadas, no teorías o especulaciones.

- **La solución no es obvia**: La mayoría de las técnicas de resolución de problemas (tales como métodos de diseño) intentan derivar soluciones partiendo de principios básicos. Los mejores patrones generan una solución para un problema indirectamente, una aproximación necesaria para los problemas de diseño más complejos.
- **Describe una relación**: Los patrones no deben describir módulos, sino que deben describir sistemas, estructuras o mecanismos más profundos.
- **El patrón debe tener un componente humano importante**: Todo software sirve para el confort humano o para la calidad de vida; los mejores patrones recurren explícitamente a la estética y a la utilidad.

Existen diferentes formatos para describir los patrones, sin embargo hay una serie de elementos que se consideran esenciales y que deben compartir todas las representaciones de los patrones, indicados por **Christopher Alexander** en [Alexander, 1979]: un patrón es una regla que establece una relación entre un **contexto**, un **sistema de fuerzas** que aparecen en el contexto y una **configuración** que permite que las fuerzas se resuelvan dentro del contexto. Posteriormente, la bibliografía especializada en patrones software ofrece diferentes variantes de los elementos considerados esenciales, pero todas ellas contienen los tres apartados indicados por el **Dr. Alexander**, que en un lenguaje más coloquial podían denominarse: *contexto, problema* y *solución*.

Así, en el libro de GoF [Gamma et al., 1995] se indica que un patrón debe contar con cuatro elementos esenciales:

1. ***El nombre del patrón****: Un descriptor manejable del problema. Debe ser corto (una palabra o dos). Amplía el vocabulario de diseño.*
2. ***El problema****: Describe cuando aplicar el patrón. Explica el problema y su contexto.*
3. ***La solución:*** *Describe los elementos que forman el diseño, sus relaciones, responsabilidades y colaboraciones. No describe un diseño o una implementación concreta, sino que ofrece una descripción abstracta de un problema.*
4. ***Las consecuencias:*** *Son los resultados de aplicar un patrón. Son necesarias para evaluar alternativas de diseño, así como para evaluar los costes y beneficios de su aplicación.*

Otros autores [Lea, 1994], [Rising, 1996] difieren en cuanto al número de elementos propios de un patrón, pero en esencia coinciden con las ideas de **Alexander** y al final vienen a representar la misma información que los patrones del libro de GoF.

Los patrones se describen utilizando formatos consistentes. Un formato es una plantilla dividida en secciones. La plantilla ofrece una uniformidad en la estructura de

los patrones que hace que éstos sean más sencillos de aprender, de comparar y de utilizar.

Existen diferentes formatos de descripción de patrones, entre los que cabe destacar el formato utilizado por Christopher Alexander en su trabajo que es denominado formato de Alexander, el formato utilizado en el libro del GoF [Gamma et al., 1995] que se denomina formato del GoF, la plantilla recomendada por **Doug Lea** (http://hillside.net/patterns/Writing/Lea.html), el formato propio del *Portland Pattern Repository* (http://www.c2.com:80/ppr/), o el formato canónico que es el utilizado en el libro POSA [Buschmann et al., 1996][29].

Debido al éxito alcanzado por el libro del GoF [Gamma et al., 1995], existe una cierta tendencia a que cada vez que se menciona el término patrón se relaciona con los patrones presentados en este libro. Dichos patrones son patrones de diseño orientado a objetos, pero existen muchos otros tipos de patrones aparte de los patrones de diseño. Se pueden citar como ejemplos de otros tipos patrones a los patrones de análisis [Fowler, 1996], a los patrones pedagógicos [Proto-Patterns, 1999], o a los patrones de organización[30], de los que existe un *website* especializado administrado por **James Coplien** (http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns). Además, los patrones enviados a las ediciones del PLoP cubren la mayoría de los campos de la Ingeniería del Software.

En [Buschmann et al., 1996] se distinguen tres tipos de patrones, marcando de una forma muy clara tres niveles conceptuales. Estos tipos son:

- **Patrones Arquitectónicos:** Expresan una organización estructural fundamental para un sistema software, que se refiere a un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellos.
- **Patrones de Diseño:** Ofrecen esquemas para refinar subsistemas y componentes de un sistema software, o las relaciones entre ellos. Describen normalmente una estructura de comunicación recurrente entre componentes, que sirve para resolver un problema general de diseño dentro de un contexto particular.
- **Idioms:** Un *idiom* es patrón de bajo nivel, específico de un determinado lenguaje de programación. Describen como implementar aspectos particulares de los componentes, o de sus interrelaciones, utilizando las características de un determinado lenguaje. Como ejemplo de un *idiom* se tiene la siguiente construcción en C++ para copiar cadenas de caracteres: `while (*destino++ = *src++);`.

[29] A los autores de este libro se les conoce popularmente como the Gang of Five (GoV).

[30] Describen las estructuras y las prácticas de las organizaciones humanas.

En [Gamma et al., 1995] se distinguen tres tipos de patrones de diseño: **patrones de creación, patrones estructurales** y **patrones de comportamiento**. Además, en cada una de las categorías se distinguen dos subtipos, patrones que trabajan con clases y patrones que trabajan con objetos.

- **Patrones de Creación:** Los patrones de creación abstraen el proceso de instanciación de objetos. Tienen la misión de permitir construir sistemas independientes de la forma en que se crean, se componen o se representan los objetos. Un patrón de creación de clases utiliza la herencia para variar la clase que es instanciada, mientras que un patrón de creación de objetos delega la instanciación en otro objeto. Como ejemplo de patrón de creación de clases se tiene el ***Factory Method***, y como ejemplo de patrón de creación de objetos se puede citar al patrón ***Builder***.
- **Patrones Estructurales:** Se cuidan de cómo las clases y los objetos se componen para formar estructuras mayores. Un patrón estructural de clases utiliza la herencia para componer interfaces o implementaciones, por ejemplo el patrón ***Adapter***. Un patrón estructural de objetos describe la forma en que se componen objetos para obtener nueva funcionalidad, además se añade la flexibilidad de cambiar la composición en tiempo de ejecución, lo cual no es posible con la composición de clases estáticas, como ejemplo de este tipo de patrones se puede mencionar al patrón ***Composite***.
- **Patrones de Comportamiento:** Este tipo de patrones está relacionado con algoritmos y la asignación de responsabilidades entre objetos. Describen, además de los patrones de objetos y clases, los patrones de comunicación entre ellos. Los patrones de comportamiento de clases utilizan la herencia para distribuir el comportamiento entre las clases, como ejemplo se puede citar al patrón ***Template Method***. Por su parte los patrones de comportamiento de objetos utilizan la composición de objetos en lugar de la herencia, por ejemplo como un grupo de objetos interconectados cooperan para realizar una tarea que un solo objeto no puede hacer por sí mismo, como representante de este tipo de patrones se puede mencionar al patrón ***Observer***.

En general se puede decir que un patrón es algo independiente de un determinado paradigma, de una metodología concreta o de un lenguaje particular. Aunque también debe señalarse que la mayor actividad en la comunidad de los patrones software ha venido de la mano de la tecnología de objetos.

La razón de por la que los patrones han tenido tan buena acogida en el seno de la Orientación al Objeto se tiene en que la Programación Orientada a Objetos se adapta muy bien a las metáforas arquitectónicas propuestas por el **Dr. Alexander**. Los objetos pueden ensamblarse en estructuras más complejas, tal y como hace la Arquitectura,

añadiendo además una dimensión temporal que recoge el intercambio de mensajes entre objetos.

Los patrones software, y en general la noción de patrón, constituyen una forma flexible y adecuada que favorece la reutilización de la experiencia embebida en los procesos que representan soluciones a problemas software, con independencia de su nivel de abstracción (análisis, diseño o implementación).

El segundo apartado se dedica a los patrones arquitectónicos. Es el tipo de patrón de mayor nivel de abstracción considerado en [Buschmann et al., 1996].

Expresan los esquemas de organización estructural fundamental de los sistemas software. Cada uno de ellos ofrece un conjunto de subsistemas predefinidos, especifica sus responsabilidades, e incluye reglas y guías para organizar sus interrelaciones. Ayudan a conseguir una propiedad global de sistema, como puede ser la adaptabilidad de la interfaz de usuario.

Los patrones arquitéctónicos se organizan en cuatro categorías:

- *Del barro a la estructura:* Los patrones en esta categoría ayudan a evitar un *mar* de componentes u objetos. En particular, soportan una descomposición controlada de una tarea de sistema en conjunto en subtareas. Esta categoría está formada por el **patrón capas** (*layers pattern*), el **patrón tuberías y filtros** (*pipes and filtres pattern*) y el **patrón pizarra** (*blackboard pattern*).
- *Sistemas distribuidos:* Incluye sólo al **patrón corredor** (*broker pattern*), haciendo referencia a dos patrones de otras categorías, el **patrón microkernel** y el **patrón tuberías y filtros**. El patrón corredor ofrece una infraestructura completa para la creación de aplicaciones distribuidas.
- *Sistemas interactivos:* En esta categoría se distinguen dos patrones, el **patrón modelo-vista-controlador** (*model-view-controller pattern*) y el **patrón presentación-abstracción-control** (*presentation-abstraction-control pattern*). Ambos patrones soportan la estructuración de los sistemas software que involucran una interacción hombre-máquina.
- *Sistemas adaptables:* Formada por dos patrones, el **patrón reflexión** (*reflection pattern*) y el **patrón microkernel**. Ambos soportan la extensión de las aplicaciones software y su adaptación a la evolución de la tecnología y el cambio de los requisitos funcionales.

El estudio individualizado de cada uno de estos patrones y su posterior combinación requeriría de unas necesidades temporales de las que no se disponen. Por ello se ha elegido uno de ellos para su presentación más detallada. Entre los citados, por su historia y su aplicación continua en las aplicaciones con interfaz gráfica de usuario, se ha elegido el patrón modelo-vista-controlador (MVC).

El patrón MVC ofrece la que puede ser la organización arquitectónica más conocida para sistemas software interactivos. Se debe a **Trygve Reenskaug** [Reenskaug et al., 1996], siendo implementada por primera vez en el entorno de Smalltalk-80 [Krasner and Pope, 1988].

Este patrón se encuentra debajo de muchos sistemas interactivos y *frameworks* de aplicación para sistemas software con interfaces gráficas, como MacApp [Apple, 1989], ET++ [Gamma, 1991], las bibliotecas de Smalltalk [Lalonde and Pugh, 1991], e incluso la biblioteca MFC de Microsoft sigue los principios del patrón MVC [Kruglinski, 1995].

El patrón arquitectónico MVC divide una aplicación interactiva en tres componentes. El modelo contiene la funcionalidad principal y los datos. Las vistas presentan la información al usuario. Los controles manejan las entradas del usuario. Las vistas y los controles representan la interfaz de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre la interfaz de usuario y el modelo.

El problema que intenta solucionar este patrón es la propensión que tienen las interfaces de usuario para cambiar, al extender la funcionalidad de una aplicación o al portarla a un entorno gráfico diferente.

Construir un sistema flexible es caro y propenso a los errores si la interfaz de usuario está altamente entremezclada con el núcleo funcional. Las *fuerzas* que intervienen en la solución son:

- La misma información es presentada de maneras diferentes en distintas ventanas.
- La presentación y el comportamiento de una aplicación deben representar los cambios en los datos inmediatamente.
- Los cambios en la interfaz de usuario deben ser sencillos e incluso factibles en tiempo de ejecución.
- Soportar diferentes estándares de interfaces gráficas o portar la interfaz de usuario no debe afectar al código del núcleo de la aplicación.

Como solución, el patrón MVC divide una aplicación interactiva en tres áreas: *el proceso, salida* y *entrada*.

El componente de modelo encapsula los datos y la funcionalidad central. El modelo es independiente de las representaciones específicas de salida o el comportamiento de entrada.

Los componentes de vista presentan la información al usuario. Una vista obtiene datos del modelo, pudiendo haber múltiples vistas del modelo.

Cada vista tiene asociado un componente controlador. Los controladores reciben la entrada, normalmente eventos que son trasladados para servir las peticiones del modelo o de la vista. El usuario interactúa con el sistema sólo a través de los controladores.

La separación del modelo de los otros dos componentes permite múltiples vistas del mismo modelo. Si el usuario cambia el modelo mediante el controlador de una vista. El modelo debe notificar a todas las vistas los cambios producidos.

Las ventajas de este patrón se pueden resumir en los siguientes puntos:

- *Múltiples vistas del mismo modelo.*
- *Vistas sincronizadas.*
- *Cambios de vistas y controles en tiempo de ejecución.*
- *Cambio del aspecto externo de las aplicaciones.*
- *Base potencial para construir un framework.*

Los inconvenientes de este patrón son los siguientes:

- *Se incrementa la complejidad.*
- *Número de actualizaciones potencialmente alto.*
- *Íntima conexión entre la vista y el controlador.*
- *Alto acoplamiento de las vistas y los controladores con respecto al modelo.*
- *Acceso ineficiente a los datos de la vista.*
- *Cambio inevitable de las vistas y controladores cuando se porte a otras plataformas.*

Los apartados tercero, cuarto y quinto presentan los diferentes tipos de patrones de diseño según [Gamma et al., 1995], esto es, patrones de creación, patrones estructurales y patrones de comportamiento, respectivamente.

Los patrones de creación se encargan de establecer mecanismos que abstraen el proceso de instanciación de los objetos. En consecuencia esta familia de patrones ofrece una gran flexibilidad en cuanto a quién, cómo, porqué y cuándo se crea un objeto.

Como sucede en todas las categorías de patrones de diseño del GoF, los patrones de creación pueden hacer referencia a clases o a objetos. Los primeros utilizan la herencia para variar la clase que va a ser instanciada, mientras que los segundos delegan la instanciación en otro objeto.

Estos patrones son especialmente interesantes en aquellas situaciones en las que priman las composiciones sobre las jerarquías de herencia. En estos casos, los sistemas se construyen definiendo pequeñas piezas con comportamiento que se van componiendo hasta formar engranajes de objetos mayores y mucho más complejos. Sobre esta idea,

los sistemas que se obtienen son configuraciones de objetos que pueden variar en estructura y funcionalidad, aumentando así considerablemente el grado de flexibilidad ofrecida por los objetos componentes.

Esta categoría de patrones cuenta con cinco representantes, de los cuales sólo uno es un patrón de creación de clases (*el patrón Factory Method*), siendo los otros cuatro restantes patrones de creación de objetos (*Abstract Factory, Builder, Prototype y Singleton*).

El patrón *Abstract Factory* también conocido como *Kit*, ofrece una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

Este patrón se aplica especialmente cuando:

- *Un sistema debe ser independiente de cómo se creen, se compongan y se representen sus objetos.*
- *Un sistema debe ser configurado con un producto de múltiples familias de productos.*
- *Una serie de productos relacionados están diseñados para usarse conjuntamente, queriéndose reforzar esta característica.*
- *Se quiere distribuir una biblioteca de clases que implementa un determinado producto, queriéndose revelar sólo sus interfaces, no sus implementaciones.*

El patrón *Builder* es otro patrón de creación de objetos, que tiene el cometido de separar la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda dar lugar a diferentes representaciones.

El uso de este patrón se adecua a situaciones como:

- *El algoritmo para crear un objeto complejo debe ser independiente de las partes que lo conforman y de la forma en que se ensamblan.*
- *El proceso de construcción debe permitir diferentes representaciones para el objeto que se construye.*

El patrón *Factory Method*, también llamado *Virtual Constructor*, es un patrón de creación de clases, esto es, define una interfaz para crear un objeto, pero deja que sean las subclases las que decidan que clase instanciar.

Su uso se adecua a situaciones en las que:

- *Una clase no puede anticipar la clase del objeto que va a crear.*
- *Una clase quiere que sus subclases especifiquen los objetos que ella crea.*

- *Las clases delegan la responsabilidad a una de varias clases auxiliares, y se quiere localizar el conocimiento de qué clase auxiliar es la escogida.*

El patrón *Prototype* (Prototipo) es un patrón de creación de objetos mediante el uso de una copia de un prototipo. El uso de este patrón se adecua a situaciones en las que un sistema debe ser independiente de la forma en que sus productos son creados, compuestos y representados.

El patrón *Singleton*, perteneciente al tipo de patrones de creación de objetos, asegura que una clase tenga una única instancia, aportando un punto de acceso global a ella.

El uso de este patrón se adecua a las situaciones en las que debe existir una sola instancia de una clase, siendo necesario que sea accesible por todos los clientes a través de una vía conocida.

Los patrones estructurales se refieren a cómo las clases y los objetos se componen para formar estructuras más grandes. Los patrones estructurales de clases utilizan la herencia para componer interfaces o implementaciones.

Por su parte los patrones estructurales de objetos describen la forma en que los objetos se componen para conseguir una nueva funcionalidad. La flexibilidad añadida de la composición de objetos viene por la habilidad de cambiar los componentes en tiempo de ejecución, lo cual es imposible con la composición estática de clases.

Esta categoría de patrones cuenta siete representantes, que se describen someramente a continuación.

El patrón *Adapter*, también conocido como *Wrapper*, convierte la interfaz de una clase en la interfaz que sus clientes esperan. Este patrón permite que clases trabajen juntas, ya que de otra forma no podrían por tener interfaces incompatibles.

Adapter es un patrón estructural tanto de clase como de objetos, que se puede aplicar cuando:

- *Se quiere utilizar clases existentes, y su interfaz no es la que se necesita.*
- *Se quiere crear una clase reutilizable que coopere con clases no relacionadas o desconocidas, esto es, clases que no tienen por que tener una interfaz compatible.*
- *Se necesita utilizar varias subclases existentes, pero no es práctico adaptar cada una de sus interfaces. Entonces, un objeto adaptador puede adaptar la interfaz de su clase padre.*

El patrón *Bridge* es un patrón estructural de objetos, también conocido como *Handle/body*, se utiliza para desacoplar una abstracción de sus implementaciones porque ambas pueden variar independientemente.

Se aplica cuando:

- *Se quiere abolir un enlace permanente entre una abstracción y su implementación. Este puede ser el caso, por ejemplo, cuando una implementación debe ser seleccionada o cambiada en tiempo de ejecución.*
- *Las abstracciones y sus implementaciones deben ser extensibles por herencia. En este caso, el patrón Bridge permite combinar diferentes abstracciones e implementaciones y extenderlas independientemente.*
- *Cambios en la implementación de una abstracción no deben tener impacto sobre sus clientes; esto es, su código no debe ser recompilado.*
- *En C++, se quiere ocultar la implementación de una abstracción completamente de sus clientes. En C++ la representación de una clase es visible en la interfaz de la clase.*
- *Se quiere compartir una implementación entre múltiples objetos, y se quiere ocultar este hecho a los clientes.*

El patrón estructural de objetos *Composite*, tiene como objetivo componer objetos en estructuras arbóreas que representen jerarquías todo/parte. Este patrón permite que los clientes traten a los objetos y a las agregaciones de objetos de una forma homogénea.

Se aplica este patrón cuando:

- *Se quieren representar jerarquías todo/parte.*
- *Se desea que los clientes ignoren la diferencia entre compuestos de objetos y objetos individuales. Los clientes deben tratar todos los objetos de un compuesto de forma uniforme.*

El patrón estructural de objetos *Decorator*, también denominado *Wrapper*, tiene como objetivo adjuntar responsabilidades adicionales a un objeto de forma dinámica. Ofrece una alternativa flexible a la herencia para extender la funcionalidad.

Se usa este patrón cuando:

- *Se quieren añadir responsabilidades a objetos individuales dinámica y transparentemente, esto es, sin afectar a otros objetos.*
- *Hay responsabilidades que deben ser retiradas.*
- *Cuando la extensión por herencia es impracticable. A veces un número alto de extensiones independientes, aunque posibles, producirían una explosión de subclases para soportar cada combinación. O una definición de clase puede estar oculta o no accesible para la herencia.*

El patrón *Facade* es un patrón estructural de objetos, que tiene por objetivo ofrecer una interfaz unificada a un conjunto de interfaces en un subsistema. Define una interfaz de alto nivel que hace al subsistema más fácil de utilizar.

Se adecua el uso de este patrón cuando:

- *Se quiere ofrecer una interfaz simple a un subsistema complejo.*
- *Hay muchas dependencias entre clientes y las clases de implementación de una abstracción. Entonces se introduce una "fachada" que desacople el subsistema de sus clientes y de otros subsistemas; esto es, promueve la independencia de los subsistemas y la portabilidad.*
- *Se desea organizar en capas los subsistemas. Se utiliza una "fachada" para definir un punto de entrada a cada subsistema. Si los subsistemas tienen dependencias, se pueden simplificas éstas haciendo que ellos se comuniquen entre sí a través de sus "fachadas".*

El patrón estructural de objetos *Flyweight* sirve para soportar un gran número de objetos de grano fino de forma eficiente.

Se aplica cuando se cumplen todas las afirmaciones siguientes:

- *Una aplicación usa un gran número de objetos.*
- *Los costes de almacenamiento son altos por la gran cantidad de objetos.*
- *La mayoría de los estados de los objetos pueden hacerse extrínsecos.*
- *Varios grupos de objetos pueden ser reemplazados por relativamente pocos objetos compartidos una vez que el estado extrínseco es eliminado.*
- *La aplicación no depende de la identidad de los objetos.*

El patrón *Proxy* es un patrón estructural de objetos que sirve para ofrecer un sustituto o un emplazamiento para otro objeto con el fin de controlar el acceso a él.

Se puede aplicar este patrón cuando:

- *Un proxy remoto ofrece una representación local a un objeto en un espacio de nombres diferente.*
- *Un proxy virtual crea objetos por demanda.*
- *Un proxy de protección controla el acceso al objeto original.*
- *Una referencia inteligente que realiza acciones adicionales cuando el objeto es accedido.*

Los patrones de comportamiento se refieren a los algoritmos y a cómo las responsabilidades se asignan entre objetos. Describen patrones de objetos y clases así como de la comunicación entre ellos. Caracterizan el flujo de control complejo que es difícil de seguir en tiempo de ejecución. Ellos desplazan la atención lejos del flujo de control para que la concentración se centre en la interconexión de los objetos.

Los patrones de comportamiento de clases utilizan la herencia para distribuir el comportamiento entre las clases. Los patrones de comportamiento de objetos utilizan la agregación en lugar de la herencia.

Esta categoría se compone de diez patrones que se resumen a continuación:

El patrón de comportamiento de objetos *Chain of Responsibility* sirve para evitar el acoplamiento entre el emisor de una petición con respecto a su receptor habilitando a más de un objeto la oportunidad de manejar la petición. Encadenar los objetos receptores y pasar la petición a través de la cadena hasta que un objeto la satisfaga.

Se usa este patrón cuando:

- *Más de un objeto puede manejar una petición, y el manejador no se conoce a priori.*
- *Se desea enviar una petición a uno de varios objetos sin especificar el receptor explícitamente.*
- *El conjunto de objetos que pueden manipular una petición debe ser especificado dinámicamente.*

El patrón de comportamiento de objetos *Command* sirve para encapsular una petición como un objeto, por ello se perametrizan los clientes con diferentes peticiones, colas o registro de peticiones, y se soportan operaciones del tipo *deshacer*.

Este patrón, que también se conoce como *Action* o *Transaction*, se aplica cuando:

- *Se quiere parametrizar objetos para realizar una acción.*
- *Se quiere especificar colas, y ejecutar peticiones en tiempos diferentes.*
- *Se desea soporte para operaciones de tipo deshacer.*
- *Se necesita soportar registro de operaciones.*
- *Se quiere estructurar un sistema alrededor de operaciones de alto nivel construidas sobre operaciones primitivas.*

El patrón de comportamiento de clases *Interpreter* sirve para que, dado un lenguaje, definir una representación para su gramática mediante un intérprete que usa la representación para interpretar las sentencias del lenguaje.

Este patrón se utiliza cuando hay un lenguaje que interpretar y se puede representar las sentencias del lenguaje como sintaxis abstracta de árboles.

El patrón de comportamiento de objetos *Iterator* o *Cursor* ofrece una forma de acceso secuencial a los elementos de un objeto agregado sin exponer su representación de fondo.

Se utiliza para:

- *Acceder a los contenidos de un objeto agregado sin exponer su representación interna.*
- *Soportar múltiples recorridos de objetos agregados.*
- *Ofrecer una interfaz uniforme para recorrer diferentes estructuras agregadas, esto es para soportar iteración polimórfica.*

El patrón de comportamiento de objetos *Mediator* define un objeto que encapsula cómo un conjunto de objetos interactúan. Este patrón proporciona un acoplamiento débil evitando que los objetos tengan que referirse los unos a los otros explícitamente, permitiendo variar sus interacciones independientemente.

Se utiliza este patrón cuando:

- *Un conjunto de objetos se comunica de formas bien definidas pero complejas.*
- *La reutilización de una clase es difícil porque se comunica con otras muchas.*
- *Un comportamiento que se distribuye entre varias clases sea configurable sin abusar de la herencia.*

El patrón de comportamiento de objetos *Memento*, también conocido como *Token*, permite, sin violar la encapsulación, capturar y exteriorizar el estado de un objeto, por lo que el estado del objeto puede ser restaurado posteriormente.

Este patrón se aplica cuando:

- *Una vista del estado de un objeto (o una porción del mismo) debe ser guardado, pudiendo ser restaurado más tarde.*
- *Una interfaz directa que obtenga el estado debería exponer detalles de implementación y romper la encapsulación del objeto.*

El patrón *Observer*, es un patrón de comportamiento de objetos, que también recibe los nombres *Dependents* o *Publish-subscribe*, define un dependencia de uno a muchos entre objetos de forma que cuando un objeto cambia, todos sus dependientes son notificados y actualizados automáticamente.

Se utiliza este patrón en cualquiera de las siguientes situaciones:

- *Cuando una abstracción tiene dos aspectos, uno dependiente del otro. Encapsulando estos aspectos en objetos separados permite variarlos y reutilizarlos independientemente.*
- *Cuando un cambio en un objeto requiere cambios en otros, y no se sabe cuantos objetos necesitan ser cambiados.*

- *Cuando un objeto debe ser capaz de notificar algo a otros objetos sin hacer presunciones sobre quiénes son esos objetos. Es decir, no se quiere tener estos objetos altamente acoplados.*

El patrón de comportamiento de objetos *State*, también denominado O*bjects for States*, permite a un objeto alterar su comportamiento cuando su estado interno cambia. El objeto parece que cambia su clase.

Se usa este patrón en cualquiera de estas situaciones:

- *El comportamiento de un objeto depende de su estado, y debe cambiar su comportamiento en tiempo de ejecución dependiendo de su estado.*
- *Las operaciones tienen sentencias grandes, con varias condiciones que dependen del estado del objeto. Este estado está normalmente representado por una o más constantes enumeradas. A menudo, varias operaciones contendrán la misma estructura condicional. El patrón State pone cada rama de la condición en una clase separada. Esto permite tratar el estado de un objeto como un objeto que puede variar independientemente de los otros objetos.*

El patrón de comportamiento de objetos *Strategy*, también conocido como *Policy*, define una familia de algoritmos, encapsulando cada uno de ellos, y haciéndolos intercambiables. Permite que el algoritmo pueda variar independientemente de los clientes que lo usan.

Se aplica este patrón cuando:

- *Varias clases relacionadas difieren sólo en su comportamiento. El patrón Strategy ofrece una forma de configurar una clase con uno de varios comportamientos.*
- *Se necesitan diferentes variantes de un algoritmo.*
- *Un algoritmo utiliza datos que sus clientes no deben conocer. Se puede emplear este patrón para evitar exponer estructuras de datos complejas y específicas de un algoritmo.*
- *Una clase define muchos comportamientos, y estos aparecen como sentencias condicionales múltiples en sus operaciones. En lugar de tener tantas condiciones, se pueden mover las ramas condicionales a una clase Strategy.*

El patrón de comportamiento de clases *Template Method* define el esqueleto de un algoritmo en un método, dejando algunos pasos para que sean definidos por las subclases. Permite, por tanto, que las subclases redefinan ciertos pasos del algoritmo sin cambiar la estructura del propio algoritmo.

Este patrón debe utilizarse cuando:

- *Se quieren implementar partes invariantes de un algoritmo una sola vez y se permite que las subclases implementen el comportamiento que puede variar.*
- *El comportamiento común entre las subclases debe ser extraído y localizado en una clase común para evitar la duplicidad de código.*
- *Se desea controlar la extensión de las subclases. Se pueden definir operaciones que se dejan incompletas (colgadas) en ciertos puntos, de forma que sólo se permiten extensiones en dichos puntos.*

El patrón de comportamiento de objetos *Visitor* representa una operación que se realiza sobre elementos de una estructura de objetos. Permite definir una nueva operación sin tener que cambiar las clases de los elementos sobre los que opera.

Se emplea este patrón cuando:

- *Una estructura de objetos está formada por muchas clases de objetos con diferentes interfaces, y se desea realizar operaciones sobre esos objetos que dependan de sus clases concretas.*
- *Varias operaciones no relacionadas deben realizarse sobre los objetos de la estructura, y no se quiere poblar las clases con dichas operaciones.*
- *Las clases que definen la estructura de objetos rara vez cambiaran, pero se desea definir nuevas operaciones sobre la estructura.*

El sexto y último apartado de este tema introduce el concepto de antipatrón. Un *antipatrón es una forma literaria que describe una solución a un problema frecuentemente utilizada que genera consecuencias negativas* [Brown et al., 1998].

Cuando un antipatrón está documentado de forma adecuada, describe de una forma general; las principales causas que conducen a la forma general; los síntomas que permiten reconocer la forma general; las consecuencias de la forma general; y una solución que describe cómo convertir el antipatrón en una solución adecuada.

Los antipatrones se caracterizan por los siguientes aspectos [Brown et al., 1998]:

- Los antipatrones son métodos para el paso eficiente de una situación general a una clase específica de soluciones.
- Los antipatrones ofrecen experiencia del mundo real para reconocer problemas recurrentes en la industria del software, además de remedios detallados para la mayoría de los aprietos más comunes.
- Los antipatrones establecen un vocabulario común para identificar problemas y discutir soluciones.
- Los antipatrones soportan una resolución sistemática a los conflictos, utilizando los recursos de la organización en todos los niveles posibles.

En [Brown et al., 1998] se distinguen los siguientes tipos de antipatrones: *antipatrones de desarrollo de software, antipatrones de arquitectura del software, antipatrones de gestión de proyectos software*.

Bibliografía

- ***Citada en las transparencias del tema:***

[Alexander, 1964] Alexander, Christopher. "*Notes on the Synthesis of Form*". Harvard University Press, 1964.

[Alexander, 1979] Alexander, Christopher. "*The Timeless Way of Building*". The Oxford University Press, 1979.

[Alexander et al., 1975] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M. and Angel, S. "*The Oregon Experiment*". Oxford University Press, 1975.

[Alexander et al., 1977] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. "*A Pattern Language*". Oxford University Press, 1977.

[Brown et al., 1998] Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". Wiley & Sons, 1998.

[Buschmann et al., 1996] Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad Peter and Stal Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

[Gamma et al., 1995] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

- ***Lecturas complementarias:***

"*History Of Patterns*". http://c2.com/cgi-bin/wiki?HistoryOfPatterns. [Última vez visitado, 27-3-2000]. March, 2000.

Datos sobre la evolución histórica de los patrones en la Orientación a Objetos.

Alexander, Christopher. "*The Origins of Pattern Theory. The Future of the Theory, and the Generation of a Living World*". IEEE Software, 16(5):71-82. September/October, 1999.

Presentación de la teoría de los patrones de la mano de su creador.

Appleton, Brad. "*Patterns and Software: Essential Concepts and Terminology*". http://www.enteract.com/~bradapp/docs/patterns-intro.html [Última vez visitado, 15-3-2000]. February, 2000.

Introducción muy completa a los patrones software. Una versión inicial de este documento se publicó en el número 5 del volumen 3 de la revista Object Magazine de mayo de 1997.

Coad, Peter. "*Object-Oriented Patterns*". Communications of the ACM, 35(9):152-159. September, 1992.

Búsqueda de patrones en el análisis y en el diseño orientado a objetos.

García Peñalvo, Francisco José. "*Patrones. De Alexander a la Tecnología de Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(10):44-52. Noviembre, 1998. (También disponible en http://tejo.usal.es/~fgarcia/doc/patrones1.pdf).

Artículo introductorio sobre los patrones software.

Kühne, Thomas. "*The Function Object Pattern*". C++ Report, 9(9):32-42. October, 1997**.**

Patrón especializado en encapsular funciones como objetos.

Lea, Doug. "*Patterns-Discussion FAQ*". http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html. [Última vez visistado, 15-3-2000]. December, 1999.

Lista de respuestas a las preguntas más frecuentes sobre patrones.

López Tallón, Alberto. "*Patrones de Diseño. Reutilización de Ideas*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(8):54-58. Septiembre, 1998.

Repasa algunos patrones definidos en el libro de GoF [Gamma et al., 1995], en concreto los patrones *Observer, Composite* y *Visitor*.

Potel, Mike. "*MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java*". Taligent, Inc. http://www.ibm.com/java/education/mvp.html [Última vez presentado, 13-3-2000]. 1996.

Variación del patrón Modelo-Vista-Controlador – MVC.

Riehle, Dirk. "*Working with Classes and Interfaces*". C++ Report, 12(3):14-21. March, 2000.

Tomando la clases como elementos fundamentales en el diseño orientado a objetos, este artículo presenta cinco patrones básicos para diseñar y utilizar clases.

Rising, Linda. "*Design Patterns: Elements of Reusable Architectures*". Annual Review of Communications, Vol. 49:907-909. 1996.

Artículo introductorio a los patrones de diseño.

Schimidt, Douglas C. "*Introduction to Design Patterns*". Washington University, St. Louis. http://www.cs.wustl.edu/~schmidt/cs242/patterns-intro4.ps.gz [Última vez visitado, 13-3-2000]. 1998.

Tutorial introductorio a los patrones de diseño.

- ***Referencias utilizadas para preparar las clases:***

a) Patrones software

"*Proceedings of the 3rd Pattern Languages of Programming Conference- PLoP'96*". Allerton Park, Illinois, Sept. 4-6, 1996. Washington University Technical Report (#wucs-97-07). Available online at http://www.cs.wustl.edu/~schmidt/PLoP-96/workshops.html. [Última vez visitado, 13-3-2000]. 1996.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the 1st European Conference on Pattern Languages of Programming - EuroPLoP '96*". Kloster Irsee, Germany, July 11-13, 1996. Washington University Technical Report (#wucs-97-07). Available online at http://www.cs.wustl.edu/~schmidt/europlop-96/writers-workshop.html. [Última vez visitado, 13-3-2000]. 1996.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the 4th Pattern Languages of Programming Conference- PLoP'97*". September 3-5, 1997 Allerton Park Monticello, Illinois, USA. Washington University Technical Report wucs-97-34. Available online at http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/proceedings.zip [Última vez visitado, 13-3-2000]. 1997.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the 2nd European Conference on Pattern Languages of Programming - EuroPLoP '97*". Siemens Technical Report 120/SW1/FB. Munich, Germany. Available online at http://www.riehle.org/events/europlop-1997/index.html [Última vez visitado, 13-3-2000]. 1997.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the 1998 Pattern Languages of Programs Conference – PLoP'98*". August 11-14, 1998 Allerton Park Monticello, Illinois, USA. Washington University Technical Report TR #WUCS-98-25. Available online at http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/ [Última vez visitado, 13-3-2000]. 1998.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the 1999 Pattern Languages of Programs Conference – PLoP'99*". August 1999. Allerton Park Monticello, Illinois, USA. Available online at http://st-www.cs.uiuc.edu/~plop/plop99/proceedings/. [Última vez visitado, 13-3-2000]. 1999.

Diversos patrones presentados en esta conferencia internacional.

"*Proceedings of the Fourth European Conference on Pattern Languages of Programming and Computing – EuroPLoP'99*". 8 - 10 July 1999. Bad Irsee, Germany. Available online at http://www.argo.be/europlop/writers.htm. [Última vez visitado, 13-3-2000]. 1999.

Diversos patrones presentados en esta conferencia internacional.

Adams, Rolf. "*Visitor or Strategy? A Comparison*". Journal of Object-Oriented Programming (JOOP), 13(1):27-28. March/April, 2000.

Estos patrones son muy similares, utilizándose para separar los algoritmos de la estructura de los objetos. Este artículo pone de manifiesto las similitudes y diferencias de los dos patrones, argumentando que la utilización del patrón *Strategy* puede utilizarse siempre en lugar del *Visitor*, dando como resultado un código más fácil de entender.

Beck, Kent. "*Patterns and Software Development. Adding Value to Reusable Software*". Dr. Dobb's Journal on CD-ROM. February, 1994.

Perpectiva del autor sobre los patrones software.

Beck, Kent, Coplien, James O., Crocker, Ron, Dominick, Lutz, Meszaros, Gerard, Paulisch, Frances and Vlissides, John. "*Industrial Experience with Design Patterns*". In Proceedings of the 18th International Conference on Software Engineering - ICSE '96. (March 25-29, 1996, Berlin, Germany). Pages 103-114. ACM, 1996.

Presenta diferentes ejemplos de utilización de los patrones de diseño en desarrollos reales. Incluye una breve historia de los patrones de diseño.

Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". Wiley & Sons, 1998.

Libro que aborda el tema de los antipatrones, presentando su concepto, los tipos, la forma de describirlos y, lo que es más importante, una gran cantidad de antipatrones de los que se pueden aprender cómo no hacer las cosas. Todo ello escrito de una forma muy amena y con gran sentido del humor.

Budinsky, M. A., Finnie, M. A., Vlissides, J. M. and Yu, P. S. "*Automatic Code Generation from Design Patterns*". IBM Systems Journal, 35(2). 1996. Also online avaliable in http://www.reasearch.ibm.com/journal/sj/budin/budinsky.html. [Última vez visitado, 12-3-2000].

Aplicación de los patrones de diseño en la generación automática de código.

Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad, Peter and Stal, Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

También conocido como POSA, junto a al libro de GoF [Gamma et al., 1995] forman la pareja de libros más importantes para introducirse en el mundo de los patrones software, y en especial en el de los patrones de diseño. Los patrones presentados se catalogan en tres grupos en función de su nivel de abstracción, yendo desde el nivel arquitectónico al nivel de implementación, dejando en medio al nivel de diseño.

Cleeland, Chris and Schmidt, Douglas C. "*External Polymorphism. An Object Structural Pattern for Transparently Extending Concrete Data Types*". In *Patterns Languages of Program Design 3*. Martin, R., Riehle, D. and Buschmann, F. editors. Pages 377-390. Addison-Wesley, 1997.

Este patrón permite que clases no relacionada por herencia y sin métodos virtuales (caso de C++) puedan ser tratadas polimórficamente, gracias a la combinación de los patrones *Adaptor* y *Decorator* [Gamma et al., 1995]. Una versión más avanzada de este patrón aparece en un artículo de título *External Polymorphism. An Object Structural Pattern for Transparently Extending C++*

Concrete Data Types, publicado en el número de septiembre de 1998 de la revista C++ Report.

Cline, Marshall P. "*The Pros and Cons of Adopting and Applying Design Patterns in the Real World*". Communications of the ACM, 39(10):47-49. October, 1996.

Ventajas y desventajas de la adopción de los patrones de diseño.

Coad, Peter, North, David and Mayfield, Mark. "*Object Models. Strategies, Patterns, & Applications*". 2nd Edition. Yourdon Press Computing Series. Prentice Hall, 1997.

Se presentan ejemplos de desarrollo de software utilizando patrones.

Coldewey, J. and Dyson, P. (editors). "*Proceedings of the Third European Conference on Pattern Languages of Programming and Computing – EuroPLoP'98*". 9 - 11 July 1998 Kloster Irsee, Germany. Available online at http://www.coldewey.com/europlop98/Program/writers.htm [Última vez visitado, 13-3-2000]. Universitätsverlag Konstanz Technicl Report. July, 1998.

Diversos patrones presentados en esta conferencia internacional.

Coplien, James O. "*Space: The Final Frontier*". C++ Report, 10(3):11-17. March, 1998.

Un artículo basado en la obra *The Nature of Order* de Christopher Alexander.

Coplien, James O. "*Baa Baa Baaa*". C++ Report, 11(9):33-37. October, 1999.

Coplien explica en este artículo cuál es el proceso de admisión y publicación de los patrones enviados a las conferencias PloP.

Coplien, James O. and Schmidt, Douglas C. (editors) "*Patterns Languages of Program Design*". Addison-Wesley, 1995.

Recoge patrones presentados en la primera conferencia sobre lenguajes de patrones PLoP.

Donadi, Mahesh. "*Rules Are for Fools, Patterns Are for Cools Fools*". Journal of Object-Oriented Programming (JOOP), 12(6):21-23,70. October, 1999.

Ejemplos, con reflexión final, de la utilización de patrones para el diseño de sistemas software.

Dodani, Mahesh. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January, 1999.

Los antipatrones como herramientas pedagógicas.

Duell, Michael. "*Non-Software Examples of Software Design Patterns*". Object Magazine, 7(5):52-57. July, 1997.

Curioso artículo que presenta ejemplos no software de los patrones de diseño del GoF.

Durán Toro, Amador. "*Apuntes de Ingeniería del software II*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla.

http://www.lsi.us.es/~amador/publicaciones.html [Última vez visitado, 14-3-2000]. 2000.

Transparencias de los temas sobre patrones de diseño.

Foster, Ted and Zhao, Liping. "*Cascade*". Journal of Object-Oriented Programming (JOOP), 11(9):18-24. February, 1999.

En este artículo se presenta el patrón *Cascade* que sirve para dividir y ordenar en capas las partes de un sistema complejo. Cada capa es un patrón *Composite* [Gamma et al., 1995] que compone los objetos en estructuras arbóreas que representan las jerarquías todo/parte.

Fowler, Martin. "*Analysis Patterns. Reusable Object Models*". Object Technology Series. Addison-Wesley, 1997.

Libro dedicado a los patrones de análisis.

Gabrilovich, Evgeniy. "*Destruction-Managed Singleton. A Compound Pattern for Reliable Deallocation of Singles*". C++ Report, 12(3):35-40,47. March, 2000.

La semántica de destrucción del patrón *Singleton* no es adecuada cuando existen varios *singletons* interrelacionados ente sí con dependencias complejas. El patrón *Destruction-Managed Singleton* completa al patrón *Singleton* imponiendo una destrucción ordenada.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

El libro de patrones de diseño por excelencia. Una obra internacionalmente reconocida, donde se presentan 23 patrones de diseño catalogados en tres familias.

Garlow, Joanne, Holmes, Chris and Mowbray, Thomas. "*Applying Design Patterns in UML*". Rose Architect Magazine, 1(2). Summer, 1999. http://www.rosearchitect.com/mag/archives/9901/f3.stml. [Última vez visitado, 15-2-2000].

Aplicación de varios patrones de diseño en el diseño e implementación de un software de comunicaciones orientado al objeto.

Gil, Joseph and Lorenz, David H. "*Design Patterns and Language Design*". IEEE Computer, 31(3):118-129. March, 1998.

Expresa las diferencias entre los patrones de diseño y los lenguajes de programación.

Goldfedder, Brandon and Rising, Linda. "*A Training Experience with Patterns*". Communications of ACM, 39(10):60-64. October, 1996.

Conceptos básicos (definiciones) sobre patrones.

Grand, Mark. "*Paterns in Java Volume 1*". John Wiley & Sons, 1998.

Presenta un catálogo de 43 patrones (11 de comportamiento, 9 estructurales, 7 de concurrencia, 6 de creación, 5 básicos y 3 de partición) desde la perspectiva de un programador de Java. En concreto dentro de este catálogo se revisan los patrones del libro de GoF [Gamma et al., 1995].

Grand, Mark. "*Paterns in Java Volume 2*". John Wiley & Sons, 1999.

Presenta un catálogo de 50 patrones (7 de resposabilidades – GRASP, 12 de interfaces gráficos de usuario, 13 de organización, 5 de optimización de código, 5 de código robusto y 8 de pruebas).

Harrison, Neil, Foote, Brian and Rohnert, Hans (editors). "*Patterns Languages of Program Design 4*". Addison-Wesley, 2000.

Recoge patrones de las últimas ediciones de las conferencias PLoP.

Helm, Richard and Gamma, Erich. "*Pattern and Software Design. Designing Objects for Extension*". Dr. Dobb's on CD-ROM. September, 1995.

Utilización de patrones para la extensión de los objetos.

Henney, Kevlin. "*Somethig for Nothing*". Java Report, 4(12):74-80. December, 1999.

Presentación del patron *Null Object*.

Hu, Weizhen. "*Singleton Pattern: Handy, but Be Careful*". Java Report, 5(3):28-32, 76. March, 2000.

Artículo que expone las limitaciones del patrón *Singleton* cuando se está trabajando en un entorno *browser* multihilo.

Kurotsuchi, Brian T. "*Welcome to the wonderful world of Design Patterns*". http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/. [Última vez visitado, 15-3-2000]. 1996.

Tutorial introductorio a los patrones de diseño.

Larman, Craig. "*The Presentation and Domain Layers*". Java Report, 4(12):64-68,82. December, 1999.

Ejemplo de utilización del patrón *GRASP Controller*.

Larman, Craig. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Introduce los patrones GRASP (General Responsibility Asignment Software Patterns – Patrones Generales de Software para Asignar Responsabilidades). Para lo que dedica los siguientes capítulos: *capítulo 18* - **GRASP: Patrones para asignar responsabilidad**; *capítulo 34* – **GRASP: Más patrones para asignar responsabilidades**; *capítulo 35* – **Diseño con más patrones**.

Larsson, Johan. "*An Introduction to Customization Design Patterns in EFLIB*". Journal of Object-Oriented Programming (JOOP), 12(5):24-28. September, 1999.

Patrones dentro de un *framework* para Object Pascal.

Lea, Doug. "*Christopher Alexander: An Introduction for Object-Oriented Designers*". ACM Software Engineering Notes, 19(1):39-46. January, 1994.

Sensacional resumen del trabajo de Alexander en la teoría de patrones, estableciendo su relación con la tecnología de objetos.

Lea, Doug. "*Design Patterns for Avionics Control Systems*". DRAFT Version 0.9.6. Technical Report DSSA Adage Project ADAGE-OSW-94-01. http://gee.cs.oswego.edu/dl/acs/acs.pdf. [Última vez visitado, 15-3-2000]. November 20, 1994.

Ejemplo de utilización de los patrones de diseño en un proyecto real.

Mannion, Mike. "*Dynamic Binder*". Java Report, 5(3):20-26. March, 2000.

Extensión de la interfaz de un objeto con nuevas operaciones y soporte de polimorfismo sin utilizar la herencia.

Martin, Robert, Riehle, Dirk and Buschmann, Frank (editors). "*Patterns Languages of Program Design 3*". Addison-Wesley, 1998.

Recoge patrones presentados en las conferencias internacionales PLoP'96 y EuroPLoP'96.

McGregor, John D. "*Test Patterns: Please Stand By*". Journal of Object-Oriented Programming (JOOP), 12(3):14-18. June, 1999.

Introducción a los patrones de pruebas.

Meszaros, Gerard and Doble, Jim. "*A Pattern Language for Pattern Writing*". http://hillside.net/patterns/Writing/pattern_index.html. [Última vez visitado, 16-3-2000]. 1996.

Técnicas para escribir patrones.

Pescio, Carlo. "*Principles Versus Patterns*". IEEE Computer, 30(9):130-131. September, 1997.

Diferencia entre principios de diseño y patrones.

Pescio, Carlo. "*Deriving Patterns from Design Principles*". Journal of Object-Oriented Programming (JOOP), 11(6):67-71. October, 1998.

Un patrón de diseño envuelve un conjunto de propiedades deseables.

Prechelt, Lutz, Unger, Barbara and Schmidt, Douglas C. "*Replication of the First Controlled Experiment on the Usefulness of Design Patterns: Detailed Description and Evaluation*". Technical Report wucs-97-34. Department of Computer Science. Washington University, St. Louis, MO 63130-4899. http://www.cs.wustl.edu/cs/techreports/1997. December, 1997.

Informe de una experiencia controlada del uso de patrones de diseño en la Universidad de Washington.

Preiss, Bruno R. "*Data Structures and Algorithms with Object-Oriented Design Patterns in C++*". John Wiley & Sons, 1999. Web Book version available at

http://www.pads.uwaterloo.ca/Bruno.Preiss/books/opus4/html/book.html. [Última vez visitado, 16-3-2000].

Libro centrado en las estructuras de datos, utilizando patrones de diseño y C++ para su diseño e implementación.

Preiss, Bruno R. "*Data Structures and Algorithms with Object-Oriented Design Patterns in Java*". John Wiley & Sons, 2000. Web Book version available at http://www.pads.uwaterloo.ca/Bruno.Preiss/books/opus5/. [Última vez visitado, 16-3-2000].

Libro centrado en las estructuras de datos, utilizando patrones de diseño y Java para su diseño e implementación.

Rising, Linda. "*The Road, Christopher Alexander, and Good Software Design*". Object Magazine. Pages 47-50. March, 1997.

Artículo que comenta la influencia de los trabajos del arquitecto Christopher Alexander sobre la teoría de patrones, en la calidad de los desarrollos software.

Rising, Linda. "*The Patterns Almanac 2000*". Addison-Wesley, 2000.

Directorio de patrones de diseño.

Schimidt, Douglas C. "*Using Design Patterns to Develop Reusable Object-Oriented Communication Software*". Communications of the ACM, 38(10). October, 1995.

Uso de los patrones de diseño en la construcción de software reutilizable en un dominio de aplicación concreto como es el sector de las comunicaciones.

Schimidt, Douglas C. "*Advanced C++ Features, Design Patterns, and Frameworks*". Washington University, St. Louis. http://www.cs.wustl.edu/~schmidt/cs242/pattern-examples4.ps.gz [Última vez visitado, 13-3-2000]. 1998.

Tutorial avanzado sobre patrones y *frameworks*.

Schmidt, Douglas C. and Stephenson, Paul. "*Using Design Patterns to Evolve System Software from UNIX to Windows NT*". C++ Report. March/April, 1995.

Artículo que presenta un uso práctico de los patrones de diseño.

Schmidt, Douglas C. and Stephenson, Paul. "*Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms*". In Proceedings of the 9th European Conference on Object-Oriented Programming – ECOOP'95. (Aarhus, Denmark on August 7-11, 1995). 1995.

Artículo que presenta un uso práctico de los patrones de diseño.

Schimidt, Douglas C., Fayad, Mohamed and Johnson, Ralph E. (guest editors). "*Software Patterns*". Communications of the ACM, 39(10):36-39. October, 1996.

Editores del especial sobre patrones software del número de octubre de 1996 de la revista *Communications of the ACM*.

Schimidt, Douglas C., Stal, Michael and Buschmann, Frank. "*Patterns for Concurrent and Distributed Objects – POSA2*". To be published by John Wiley & Sons in 2000.

Many drafts are available at http://www.cs.wustl.edu/~schmidt/patterns/patterns.html [Última vez visitado, 13-3-2000]. 2000.

Borradores de algunos de los patrones que se incluirán en un próximo libro, denominado por los autores como POSA2. Los patrones son fruto del proyecto de investigación JOLT entre *Siemens Corporate Technology* y la *Washington University*. Son patrones dirigidos hacia aplicaciones distribuidas.

Vlissides, John. "*Pattern Hatching. Design Patterns Applied*". Software Patterns Series. Addison-Wesley, 1998.

John Vlissides, uno de los autores del prestigioso libro de GoF [Gamma et al., 1995] y autor de una columna bimensual sobre patrones en la revista *C++ Report*, presenta en este libro la experiencia de la aplicación y la creación de patrones de diseño, exponiendo de forma concisa variaciones a algunos de los patrones del libro de GoF, su aplicación en desarrollos reales y consejos para la creación de nuevos patrones.

Vlissides, John and Alexandrescu, Andrei. "*To Code or Not to Code, Part 1*". C++ Report, 12(3):44-47. March, 2000.

Sobre la aplicación de los patrones en la generación automática de código.

Vlissides, J., Coplien, J. and Kerth, N. (editors) "*Patterns Languages of Programming Design 2*". Addison-Wesley, 1996.

Recoge patrones presentados en la segunda edición del PLoP.

b) Frameworks

Aklecha, Vishwajit. "*Object-Oriented Frameworks Using C++ and CORBA Gold Book*". Coriolis Technology Press, 1999.

El libro en su totalidad es de sumo interés en cuanto a su relación con los *frameworks*. Por su carácter introductorio se destacan los capítulos 5, **Introduction to design patterns**, y 6, **Concepts of frameworks**.

Bäumer, Dirk, Gryczan, Guido, Knoll, Rolf, Lilienthal, Carola, Riehle, Dirk and Züllighoven, Heinz. "*Framework Development for Large Systems*". Communications of the ACM, 40(10):52-59. October, 1997.

Artículo que se deriva de la experiencia de los autores en el desarrollo de aplicaciones en el sector bancario.

Carlson, Brent. "*Design Patterns for Business Frameworks and Applications*". Java Report, 5(3):34-46. March, 2000.

Artículo que indica la potencia de los patrones de diseño cuando se usan de una forma concertada. Cuando se utilizan para la creación de un *framework* éstos describen las interacciones entre varios objetos del negocio de una forma genérica y consistente, de forma que el desarrollador aprende rápidamente cómo usar y extender el *framework*.

Codenie, Wim, Hondt, Koen De, Steyaert, Patrick and Vercammen, Arlette. "*From Custom Applications to Domain-Specific Frameworks*". Communications of the ACM, 40(10):71-77. October, 1997.

Justificación de los *frameworks* para dominios verticales.

Demeyer, Serge. "*Zypher: Tailorability as a Link from Object-Oriented Software Engineering to Open Hypermedia*". Phd dissertation. Faculty of Sciences, Brussels Free University - July, 1996. http://dinf.vub.ac.be/~demeyer/Zypher/zpprpdf/ [Última vez visitado, 13-3-2000]. Last revision: January, 29th 1997.

En su capítulo 2, **Object-Oriented Software Engineering**, trata el tema de los *frameworks*.

Demeyer, Serge, Meijler, Theo Dirk, Nierstrasz and Steyaert, Patrick. "*Design Guidelines for Tailorable Frameworks*". Communications of the ACM, 40(10):60-64. October, 1997.

La interoperabilidad, la extensibilidad y la distribución son las tres guías que estos autores proponen para la creación de *frameworks*.

Fayad, Mohamed E. and Schmidt, Douglas C. (guest editors) "*Object-Oriented Application Frameworks*". Communications of the ACM, 40(10):32-38. October, 1997.

Estos dos importantes personajes dentro de la OO son los editores del especial sobre *frameworks* del número de octubre de 1997 de la revista *Communications of the ACM*, realizando ellos mismos una correcta introducción a los *frameworks*.

Johnson, Ralph E. "*Documenting Frameworks Using Patterns*". In Conference Proceedings on Object-oriented Programming Systems, Languages, and Applications – OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 63-76. ACM, 1992.

Artículo que utiliza los patrones para la documentación de los *frameworks*.

Johnson, Ralph E. "*Frameworks = (Components + Patterns)*". Communications of the ACM, 40(10):39-42. October, 1997.

Visión de los *frameworks* desde el punto de vista de este autor.

Larsen, Grant. "*Designing Component-Based Frameworks Using Patterns in the UML*". Communications of the ACM, 42(10):38-45. October, 1999.

Aproximación a la creación de *frameworks*.

Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

Este libro contiene dos capítulos relacionados con los patrones y los frameworks. El capítulo 5, **Creating reusable components**, se introduce el concepto de patrón y de *framework*. El capítulo 9, **Case study: the creation of a framework**, explica como construir un *framework* utilizando el enfoque OOram y su documentación con patrones.

Richards, Ben H. "*Frameworks and Design Patterns*". MultiUse Express, 5(1):2. February, 1997.

Artículo introductorio que define lo que es un *framework* y un patrón.

Schmid, Hans Albrecht. "*Systematic Framework Design by Generalization*". Communications of the ACM, 40(10):48-51. October, 1997.

Artículo sobre el diseño de los puntos de variabilidad en los *frameworks*.

Schmidt, Douglas C. and Fayad, Mohamed E.. "*Lessons Learned. Building Reusable OO Frameworks for Distributed Software*". Communications of the ACM, 40(10):85-87. October, 1997.

Diseño de *frameworks* para aplicaciones distribuidos.

Taligent, Inc. "*Leveraging Object-Oriented Frameworks*". Taligent White Papers. IBM. http://www.ibm.com/java/education/ooleveraging/index.html [Última vez visitado, 13-3-2000]. 1993.

Aproximación de Taligent al desarrollo de *frameworks*.

Taligent, Inc. "*Building Object-Oriented Frameworks*". Taligent White Papers. IBM. http://www.ibm.com/java/education/oobuilding/index.html [Última vez visitado, 13-3-2000]. 1994.

Introducción al desarrollo de *frameworks*.

Wirfs-Brock, Allen, Vlissides, John, Cunningham, Ward, Johnson, Ralph and Bollette, Lonnie. "*Panel: Designing Reusable Designs: Experiences Designing Object-Oriented Frameworks*". In Addendum to the Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA90/ECOOP90. (October 21 - 25, 1990, Ottawa Canada). Pages 19-24. ACM, 1990.

Presentación de los *frameworks* orientados a objetos como uno de los elementos más representativos del desarrollo para reutilización en la fase de diseño.

Yacoub, Sherif M. and Ammar, H. H. "*Towards Pattern-Oriented Frameworks*". Journal of Object-Oriented Programming (JOOP), 12(8):25-38,46. January, 2000.

Se presenta en este artículo una aproximación para construir *frameworks* de diseño OO utilizando patrones como elementos de construcción.

c) Arquitecturas software

Cockburn, Alistair. "*The Interaction of Social Issues and Software Architecture*". Communications of the ACM, 39(10):40-46. October, 1996.

A la hora de diseñar las arquitecturas se deben tener en cuenta factores humanos y técnicos.

Coplien, James O. "*Idioms and Patterns as Architectural Literature*". IEEE Software, 14(1):36-42. January, 1997.

Relaciones entre los conceptos de objeto, patrón y arquitectura.

Kerth, Norman L. and Cunningham, Ward. "*Using Patterns to Improve Our Architectural Vision*". IEEE Software, 14(1):53-59. January, 1997.

Importancia de los lenguajes de patrones en la definición de una arquitectura software.

Monroe, Robert T., Kompanek, Andrew, Melton, Ralph and Garlan, David. "*Architectural Styles, Design Patterns, and Objects*". IEEE Software, 14(1):43-52. January, 1997.

Relación entre los estilos arquitectónicos en el desarrollo de sistemas software y los patrones de diseño en la Orientación a Objetos.

Shaw, Mary and Garlan, David. "*Software Architecture. Perspectives on an Emerging Discipline*". Prentice Hall, 1996.

Cuando el tamaño y la complejidad de los sistemas software crece, el diseño y especificación de la estructura global del sistema se convierte en un asunto más significativo que la propia elección de los algoritmos y las estructuras de datos. Este libro presenta la importancia del diseño del nivel arquitectónico del software, siendo una de las referencias clave en la disciplina que se conoce como *Arquitecturas Software*.

Tepfenhart, William M. and Cusick, James J. "*A Unified Object Topology*". IEEE Software, 14(1):31-35. January, 1997.

Ofrecen una topología donde catalogan los términos relacionadas con la arquitectura software, los *frameworks*, los patrones y los objetos, para aclarar donde hacer énfasis dependiendo de la aplicación a construir.

Tema 7: *Diseño por Contrato*

Descriptores

Diseño por contrato; programación por contrato; corrección del software; fiabilidad; contrato; aserción; precondición; postcondición; invariante.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T7, T9** y **P2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.
- Estudio y comprensión de los fundamentos del diseño de sistemas software.
- Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar como parte del método, la utilización de aserciones, a fin de lograr corrección y fiabilidad en los diseños software realizados.
- Ver el diseño por contrato como una filosofía fundamental dentro de la reutilización en la Orientación a Objetos.
- Establecer el diseño por contrato como el desencadenante de las acciones a tomar en caso de incumplimiento del contrato (errores, excepciones).

Contenidos

7.1 Introducción al diseño por contrato
7.2 Aserciones: Pre y postcondiciones
7.3 Invariantes de clase
7.4 Aserciones en C++

Tabla 5.31. Contenidos del séptimo tema de teoría de Programación Orientada a Objetos

Resumen

Uno de los factores de calidad que se deben pedir a un sistema software es su corrección, lo cual se hace más imperioso, si cabe, cuando la reutilización es uno de las metas que se persiguen.

Cuando un módulo se va a utilizar en contextos para los que inicialmente no fue diseñado, las posibilidades de descubrir un error oculto se hacen mayores. Un ejemplo reciente de esto fue el primer lanzamiento del **Ariane 5**, que falló *explosivamente* por la reutilización incorrecta de una rutina del sistema de control de vuelo del **Ariane 4** [Jézéquel and Meyer, 1997].

Desde esta perspectiva se introducen las técnicas de diseño bajo contrato, que garanticen de alguna forma la corrección del software, y que constituyen el contenido

del último tema del programa teórico de Programación Orientada a Objetos. Para ello se ha dividido el tema en cuatro apartados principales que se introducen a continuación.

El primer apartado sirve para introducir el diseño por contrato [Meyer, 1997], haciendo hincapié en la importancia de a corrección del software.

Los mecanismos del diseño por contrato, aunque no son infalibles, proporcionan al ingeniero del software las herramientas esenciales para expresar y verificar argumentos sobre la corrección.

El diseño por contrato ve las relaciones entre clases y sus clientes como un acuerdo formal, que expresa los derechos y obligaciones de cada parte. Sólo a través de una definición precisa de las afirmaciones y responsabilidades de cada módulo se puede aspirar a alcanzar un grado significativo de confianza en los grandes sistemas [Meyer, 1997].

La corrección del software es un concepto relativo. Un sistema software o un elemento software no puede decirse que sea correcto o incorrecto *per se*; es correcto o incorrecto con respecto a cierta especificación. Estrictamente hablando, no se debería discutir sobre si un sistema software es o no correcto, sino sobre si es consistente con sus especificaciones.

El paradigma del diseño por contrato expresa que: "*Si el cliente (el que envía un mensaje) cumple ciertos requisitos cuando usa al servidor, entonces el servidor (la función llamada) garantiza que se produce un resultado exitoso. El beneficio para el cliente es que se conoce que el servidor siempre ofrece un resultado válido, siempre que él satisfaga su parte del contrato. El servidor se beneficia del conocimiento de que sólo necesita producir un resultado cuando el cliente lo invoca adecuadamente (de acuerdo a los términos del contrato). Cuando los términos del contrato son explícitos, las obligaciones para el chequeo de errores son claras*" [Meyer, 1997].

El segundo apartado introduce el concepto de aserción como a herramienta que permite trasladar la especificación al código de los métodos.

Como elemento previo se repasan los conceptos básicos para la expresión de una especificación utilizando fórmulas de corrección, también denominadas *Tripletas de Hoare*. Así, si `A` es una cierta operación, una fórmula de corrección es una expresión de la forma:

```
{P} A {Q}
```

que denota la propiedad siguiente, que puede ser cierta o no:

"*Una ejecución de* `A` *que comience en un estado en el que se cumpla* `P` *terminará en un estado en el que se cumple* `Q`" [Meyer, 1997].

Las fórmulas de corrección son una notación matemática, no una construcción de programa; no son parte del lenguaje de programación.

En `{P}A{Q}`, `A` denota una operación; mientras que `P` y `Q` son propiedades de las diferentes entidades implicadas. Estas propiedades reciben el nombre de aserciones, donde `P` se denomina precondición y `Q` postcondición.

Una aserción es "*una expresión que involucra algunas entidades software y que establece una propiedad que dichas entidades deben satisfacer en ciertas etapas de la ejecución del software*" [Meyer, 1997].

Matemáticamente la noción más cercada a la de aserción es la de predicado, aunque, por ejemplo en Eiffel, el lenguaje de aserciones que se utiliza tiene sólo parte de la potencia del cálculo de predicados completo.

Se puede especificar la tarea que lleva a cabo una rutina mediante dos aserciones asociadas a la rutina: una *precondición* y una *postcondición*. La precondición establece las propiedades que se tienen que cumplir cada vez que se llame a la rutina; la postcondición establece las propiedades que debe garantizar la rutina cuando retorne.

Definir una precondición y una postcondición para una rutina es una forma de definir un *contrato* que liga a la rutina con quienes la llaman. De manera que si un método *`r`* de la clase *`C`* tiene cláusulas *`P`* y *`Q`*, se puede interpretar que la clase dice a sus clientes: *Si me garantizas llamar a `r` con `P` cierta, yo te garantizo que te devolveré un estado final en el que `Q` también sea cierta*.

Siguiendo la idea de los contratos, cuando un contrato es quebrantado por una de las dos partes, la otra no tiene porqué sentirse obligada por él. Así, si la precondición de un método no es satisfecha por el cliente que solicita su ejecución, el método no tiene porqué garantizar ningún estado final, más bien debería *denunciar* a su cliente por incumplimiento de contrato.

Las aserciones no son sistemas de chequeo de las entradas de datos, ni estructuras de control.

El tercer apartado se dedica al concepto de invariante. La técnica de especificación utilizando precondiciones y postcondiciones está especialmente diseñada para ligar con condiciones a los métodos, pero falla cuando se intenta expresar propiedades generales de toda la clase, que deben ser respetadas por todos los métodos.

El invariante de una clase se verifica en los estados estables de la clase. Parece evidente que la clase necesita incumplir sus invariantes para pasar de un estado que los verifica a otro estado en las mismas condiciones. Las situaciones inestables de la clase se pueden resumir en:

- En el interior de los métodos de la clase.
- En las llamadas no calificadas a la clase.

La siguiente regla define con precisión si una aserción es o no un invariante correcto para una clase [Meyer, 1997]: *Una aserción `I` es un invariante correcto de la clase `C` si y sólo si cumple las dos condiciones siguientes:*

1. *Todo procedimiento de creación de `C`, cuando se aplica a argumentos que satisfacen su precondición en un estado donde los atributos tienen sus valores por defecto, produce un estado que satisface `I`.*
2. *Toda rutina exportada de la clase, cuando se aplica a argumentos y en un estado que satisface tanto a `I` como a la precondición de la rutina, produce un estado que satisface a `I`.*

Los invariantes tienen una interpretación clara en la metáfora del contrato. Los contratos humanos suelen tener alusiones a cláusulas o regulaciones generales que se aplican a todos los contratos de una cierta categoría. Los invariantes juegan un papel similar en los contratos software: el invariante de una clase afecta a todos los contratos entre una rutina de la clase y un cliente.

El invariante se puede considerar como algo añadido a toda precondición y a toda postcondición de toda rutina exportada. Sea $cuerpo$ el cuerpo de la rutina, pre su precondición, $post$ su postcondición e INV el invariante de la clase. El requisito de corrección de una rutina se puede expresar en la forma:

$$\{INV \wedge pre\}\ cuerpo\ \{INV \wedge post\}$$

Con las precondiciones, las postcondiciones y los invariantes, se puede definir lo que significa que una clase sea correcta.

Una clase, como cualquier otro elemento software, es correcto o incorrecta no por sí misma sino con respecto a una especificación. Introduciendo las precondiciones, postcondiciones e invariantes se tiene un mecanismo para incluir algo de las especificaciones en el propio texto de la clase.

Sea C una clase, INV el invariante de la clase. Para cada rutina r de la clase, se llamará $pre_r(x_r)$ y $post_r(x_r)$ a su precondición y su postcondición respectivamente; donde x_r denota a los posibles argumentos de r, a los cuales se pueden referir tanto la precondición como la postcondición. Se denomina $cuerpo_r$ al cuerpo de la rutina r.

Por último, sea $Por\_defecto_C$ la aserción que expresa que los atributos de C tienen los valores por defecto de sus tipos respectivos.

Con estas notaciones se puede dar una definición general de corrección [Meyer, 1997]: *Una clase es correcta con respecto a sus aserciones si y sólo si:*

1. *Para cada conjunto válido de argumentos x_p de un procedimiento de creación p:* $\{Por\_defecto_C \wedge pre_p(x_p)\}\ cuerpo_p\ \{post_p(x_p) \wedge INV\}$.

2. *Para cada rutina exportada* r *y cualquier conjunto* x_r *de argumentos válidos:* $\{pre_r(x_r) \wedge INV\}\ cuerpo_r\ \{post_r(x_r) \wedge INV\}$.

El cuarto y último apartado del presente tema estudia la aplicación del diseño por contrato en C++.

C++ tiene unos mecanismos bastante limitados para el soporte de las aserciones, sin embargo en [Welch and Strong, 1998] presentan una forma de definir una mecanismo de aserciones más completo para C++, sirviendo el ejemplo que se desarrolla en este artículo como base para el desarrollo de este apartado.

Bibliografía

- ***Citada en las transparencias del tema:***

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Welch and Strong, 1998] Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

- ***Lecturas complementarias:***

Interactive Software Engineering. "*Building Bug-Free O-O Software: An Introduction to Design by Contract*". Interactive Software Engineering. http://www.eiffel.com/doc/manuals/technology/contract/page.html. [Última vez visitado, 12-3-2000]. 1999.

Introducción a los aspectos básicos del diseño por contrato..

Jézéquel, Jean-Marc and Meyer, Bertrand. "*Design by Contract: The Lessons of Ariane*". IEEE Computer, 30(1):129-130. January, 1997.

Artículo que, incidiendo en el fallo del Ariane 5, vincula al diseño por contrato como elemento necesario para el desarrollo de módulos reutilizables. Una versión de este artículo está accesible en línea en la URL siguiente: http://www.eiffel.com/doc/manuals/technology/contract/ariane/page.html.

Meyer, Bertrand. "*Applying 'Design by Contract'*". IEEE Computer, 25(10):40-51. October, 1992.

Artículo en el que Meyer introduce el Diseño por Contrato.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

El capítulo 11, **Diseño por contrato: construcción de software fiable**, introducción del paradigma de diseño por contrato dentro de la Orientación a Objeto. Utiliza el soporte que da el lenguaje Eiffel para expresar los ejemplos del capítulo.

- ***Referencias utilizadas para preparar las clases:***

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Dentro de su capítulo 13, **Projects, Design, and Other Factors**, se tiene un apartado dedicado al diseño por contrato.

Moros Valle, Begoña, Nicolás Ros, Joaquín, García Molina, Jesús and Toval Álvarez, José Ambrosio. "*Combining Formal Specifications with Design by Contract*". Journal of Object-Oriented Programming, 12(9):16-21,46. February, 2000.

Propuesta de desarrollo de software orientado al objeto basado en generación automática de un prototipo partiendo de especificaciones formales, validación y refinamiento de los requisitos mediante el prototipo y conversión automática de las especificaciones de los tipos validados a clases que incluyen la semántica de las especificaciones formales utilizando aserciones.

Norris, Eugene N. "*Lecture Notes of the CS 699-001 – Object-Oriented Design and Implementation in C++ and Java*". http://www.cs.gmu.edu/~Eenorris/O-Ocourses/699Syllabus.html. Spring 1998.

Apuntes de clase de este curso. Pueden utilizarse para otros temas, pero en relación con el Diseño por Contrato se destaca su capítulo 6 **Design by Contract**.

Prieto Arambillet, Félix. "*Apuntes de la Asignatura Programación III*". Versión 1.0. Departamento de Informática. Universidad de Valladolid. Diciembre, 1999.

Uno de los temas está dedicado por completo a la *programación bajo contrato*.

Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

Artículo para establecer un mecanismo de aserciones en C++.

5.3.1.3 Bibliografía empleada en la parte teórica de la asignatura

En este apartado se va a citar todas las referencias utilizadas para impartir la parte de teoría de la asignatura Programación Orientada a Objetos. Para ello se van a distinguir los tres apartados que se han venido presentando en el desarrollo comentado de cada uno de los temas de este programa: *las referencias citadas en las transparencias, las lecturas complementarias* y *las referencias para preparar las clases.*

Bibliografía citada en las transparencias

Abbott, R. J. "*Program Design by Informal English Descriptions*". Communications of the ACM, 26(11):882-894. November, 1983.

Ader, M., Nierstrasz, O., McMahon, S., Muller, G. and Pröfrock, A.-K. "*The ITHACA Technology: A Landscape for Object-Oriented Application Development*". In Proceedings of ESPRIT'90 Conference. Kluwer Academic Publisher. 1990.

Alexander, Christopher. "*Notes on the Synthesis of Form*". Harvard University Press, 1964.

Alexander, Christopher. "*The Timeless Way of Building*". The Oxford University Press, 1979.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M. and Angel, S. "*The Oregon Experiment*". Oxford University Press, 1975**.**

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. "*A Pattern Language*". Oxford University Press, 1977.

Ancona, D., Astesiano, E. and Zucca, E. "*Towards a Classification of Inheritance Relations*". Technical Report, Dipartimento di Informatica e Scienze dell'Informazione, Genova. 1992.

Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

Biggerstaff, T. J. "*Design Recovery for Maintenance and Reuse*". IEEE Computer, 22(7):36-49. July, 1989.

Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

Brown, A. W. and Wallnau, K. C. "*The Current State of CBSE*". IEEE Software, 15(5):37-46. September-October, 1998.

Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". John Wiley & Sons, 1998.

Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad Peter and Stal Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Cardelli, L. and Wegner, P. "*On Understanding Types, Data Abstraction and Polymorphism*". ACM Computing Surveys, 17(4). 1985.

Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

Do Prado Leite, J. C. S., Sant'Anna, M. and de Freitas, F. G. "*Draco-PUC: A Technology Assembly for Domain Oriented Software Development*". In Proceedings of the Third International Conference on Software Reusability ICSR-3. W. B. Frakes editor. (Rio de Janeiro, Brazil, 1-4 November 1994). Pages 102-109. IEEE Press, 1994.

DoD. "*DoD Software Reuse Vision and Strategy*". Technical Report 1222-04-210/40, Department of Defense (DoD Software Software Reuse Initiative), Falls Church, VA. 1992.

Firesmith, Donald, Henderson-Sellers, Brian and Graham, Ian. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

Girardi, M. Rosario. "*Main Approaches to Software Classification and Retrieval*". En las actas del curso *Ingeniería del Software y Reutilización: Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

Goldberg, Adele and Robson, David. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

Griss, Martin L. "*Software Reuse: From Library to Factory*". IBM System Journal, 32(4):1-23. November, 1993.

Griss, M. L. and Wentzel, K. D. "*Hybrid Reuse with Domain-Specific Kits*". In Paulin, J. and Tracz, W. editors, WISR'93: 6th Annual Workshop on Software Reuse. Summary and Working Group Reports. 1993.

Halbert, D. and O'Brien, P. "*Using Types and Inheritance in Object-Oriented Programs*". IEEE Software, pages 71-79. 1987.

Jacobson, Ivar. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 1: Languages*". Journal of Object-Oriented Programming, 11(8):8-15. January, 1999.

Kruchten, P., Schonberg, E. and Schwartz, J. T. "*Software Prototyping Using the SETL Programming Language*". IEEE Software, 1(4):66-75. 1984.

Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

Liskov, Barbara. "*Data Abstraction and Hierarchy*". In Addedum to Proceedings of OOPSLA'87. Pages 17-35. ACM Press, 1987.

Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

Marqués Corral, José Manuel. "*Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación*". En las actas del curso *Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

McClure, Carma. "*Software Reuse Techniques: Adding Reuse to the Systema Development Process*". Prentice-Hall, 1997.

McIlroy, Doug. "*Mass-Produced Software Components*". In *Software Engineering Concepts and Techniques; 1968 NATO Conference on Software Engineering*. J. M. Buxton, P. Naur and B. Randell editors. Pages 88-98. Van Nostrand Reinhold, 1976.

Meyer, Bertrand. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

NATO. "*NATO Standard for Management of a Reusable Software Component Library*". Volume 2 (of 3 Documents). NATO Communications and Information Systems Agency (NACISA). 1992.

Neighbors, J. M. "*The Draco Approach to Constructing Software from Reusable Components*". IEEE Transactions on Software Engineering, SE-10(5):564-574. September, 1984.

Oestereich, Bernd. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Pastor, Óscar and Ramos, Isidro. "*OASIS Version 2 (2.2): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*". Servicio de publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-95.788. 1995.

Peterson, S. A. "*Coming to Terms with Software Reuse Terminology: A Model-Based*". ACM Software Engineering Notes, 16(2):45-51. April, 1991.

Prieto-Díaz, Rubén. "*Classification of Reusable Modules*". IEEE Software 4(1):6-16, January, 1987.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

Sakkinen, M. "*Disciplined Inheritance*". In Proceedings of ECOOP'89. Cook, S. editor. Pages 39-56. Cambridge University Press, 1989.

SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

STARS. "*STARS Conceptual Framework for Reuse Processes (CFRP)*". Technical Report STARS-VC-A018/001/00, STARS. Version 3.0 Vols I & II. 1993.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". Technical Report HPL-94-34, Hewlett-Packard Laboratories, April 1994.

Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

Tesler, L. "*The Smalltalk Environment*". Byte, 6(8). August, 1981.

Tesler, L. "*Object-Oriented Dynamic Languages*". In Proceedings of the Object Expo Conference. July, 1993.

Tracz, Will. "*Confessions of a Used Program Salesman: Institutionalizing Software Reuse*". Addison-Wesley, 1995.

Voas, J. M. "*The Challenges of Using COTS Software in Component-Based Development*". IEEE Computer, 31(6):44-45. June, 1998.

Wallnau, K. C. "*Towards an Extended View of Reuse Libraries*". In Proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR-5), Palo Alto, California (USA). 1992.

Wegner, Peter. "*The Object-Oriented Classification Paradigm in Research Directions on Object-Oriented Programming*". MIT Press, Cambridge, MA, 1987.

Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

Lecturas complementarias

"*History Of Patterns*". http://c2.com/cgi-bin/wiki?HistoryOfPatterns. [Última vez visitado, 27-3-2000]. March, 2000.

"*The Real Stroustrup Interview*". IEEE Computer, 31(6):110-114. June, 1998.

Adolph, Steve. "*Whatever Happened to Reuse?*". Software Development. http://www.sdmagazine.com/breakrm/features/s9911f3.shtml [Última vez visitado, 24-2-2000]. November, 1999.

Alexander, Christopher. "*The Origins of Pattern Theory. The Future of the Theory, and the Generation of a Living World*". IEEE Software, 16(5):71-82. September/October, 1999.

Appleton, Brad. "*Patterns and Software: Essential Concepts and Terminology*". http://www.enteract.com/~bradapp/docs/patterns-intro.html [Última vez visitado, 15-3-2000]. February, 2000.

Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

Binder, Robert. "*Verifying Class Associations*". Object Magazine, 7(9):18-20. November, 1997.

Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". Wiley & Sons, 1998.

Cardelli, Luca and Wegner, Peter "*On Understanding Types, Data Abstraction and Polymorphism*". Computing Surveys, 17(4):471-523. December, 1985.

Coad, Peter. "*Object-Oriented Patterns*". Communications of the ACM, 35(9):152-159. September, 1992.

Devis Botella, Ricardo. "*Contenedores y Plantillas en C++*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(5):61-71. Marzo, 1995.

Devis Botella, Ricardo. "*Manejo de Excepciones en C++*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(2):45-53. Febrero, 1996.

DoD. "*Software Reuse Executive Primer*". Produced by DoD Software Reuse Initiative. April, 1996.

Fernández Sánchez, José Luis. "*Reusabilidad y Desarrollo Orientado a Objetos*". En las actas de las Segundas Jornadas sobre Tecnología de Objetos. Madrid, noviembre, 1996. http://www.ati.es/GRUP_TRABAJO/LATIGOO/OOp96/Ponen7/atio6p07.html. [Última vez visitado, 29-2-2000]. 1996.

García Peñalvo, Francisco José. "*Patrones. De Alexander a la Tecnología de Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(10):44-52. Noviembre, 1998. (También disponible en http://tejo.usal.es/~fgarcia/doc/patrones1.pdf).

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*El Principio Abierto/Cerrado*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(6):52-56. Junio, 1998.

García Peñalvo, Francisco José y Marqués Corral, José Manuel. "*El Principio de Sustitución de Liskov*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(7):40-44. Julio, 1998.

García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre, 1997.

Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

Henderson-Sellers, B. "*OPEN Relationships- Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December, 1997.

Interactive Software Engineering. "*Building Bug-Free O-O Software: An Introduction to Design by Contract*". Interactive Software Engineering. http://www.eiffel.com/doc/manuals/technology/contract/page.html. [Última vez visitado, 12-3-2000]. 1999.

Interactive Software Engineering. "*Object-Oriented Languages: A Comparison*". Interactive Software Engineering (ISE). http://www.eiffel.com/doc/manuals/technology/oo_comparison/index.html [Última vez visitado, 24-2-2000]. 1999.

Jézéquel, Jean-Marc and Meyer, Bertrand. "*Design by Contract: The Lessons of Ariane*". IEEE Computer, 30(1):129-130. January, 1997.

Joyner, Ian. "*C++?? A Critique of C++ and Programming and Language Trends of the 1990s*". 3rd edition http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/cppcv3.pdf. [Última vez visitado 7/1/2000]. 1996.

Kay, Alan C. "*The Early History of Smalltalk*". In Proceedings of the second ACM SIGPLAN conference on History of programming languages – HOPL II. (April 20 - 23, 1993, Cambridge United States). ACM SIGPLAN Notices, 28(3):69-75. March, 1993.

Kirman, Jak. "*A Modest STL Tutorial*". http://www.cs.brown.edu/people/jak/proglan/stltut/tut.html. [Última vez visitado, 10-11-1999]. January 1998.

Lea, Doug. "*Patterns-Discussion FAQ*". http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html. [Última vez visistado, 15-3-2000]. December, 1999.

López Tallón, Alberto. "*Patrones de Diseño. Reutilización de Ideas*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(8):54-58. Septiembre, 1998.

Meyer, Bertrand. "*Applying 'Design by Contract'*". IEEE Computer, 25(10):40-51. October, 1992.

Meyer, Bertrand. "*Approaches to Portability*". Journal of Object-Oriented Programming (JOOP), 11(4):68-70. July/August, 1998.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Meyer, Bertrand. "*The Significance of Components*". Software Development. http://www.sdmagazine.com/uml/beyondobjects/s9911bo1.shtml [Última vez visitado, 24-2-2000]. November, 1999.

Meyer, Bertrand. "*What to Compose*". Software Development. http://www.sdmagazine.com/uml/beyondobjects/s0003bo.shtml [Última vez visitado, 12-3-2000]. March, 2000.

Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". In Proceedings of the First International Joint Conference of ISSAC-88 and AAECC-6. P. Gianni editor. (Rome, Italy, July 4-8, 1988). Published in *Lecture Notes in Computer Science* 358, Pages 13-25. Springer-Verlag, 1989.

Musser, David R. and Stepanov, Alexander A. "*Algorithm-Oriented Generic Library*". Software Practice & Experience, 24(7):623-642. July, 1994.

Parsons, Jeffrey and Wand, Yair. "*Choosing Classes in Conceptual Modeling*". Communications of the ACM, 40(6):63-69. June, 1997.

Piattini Velthuis, Mario Gerardo. "*Selección de Lenguajes de Programación Orientados al Objeto: ¿Cuestión de Religión?*". Revista BASE de la ALI (Asociación de Doctores, Licenciados e Ingenieros en Informática), Nº24:58-62. Abril, 1994.

Potel, Mike. "*MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java*". Taligent, Inc. http://www.ibm.com/java/education/mvp.html [Última vez presentado, 13-3-2000]. 1996.

Prechelt, Lutz. "*Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences*". Communications of the ACM, 42(10):109-112. October, 1999.

Pressman, Roger S. "*Ingeniería del Software. Un Enfoque Práctico*". 4ª Edición. McGraw-Hill, 1998.

Rans, Michael. "*A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development*". http://users.ox.ac.uk/~ball0370/documents/oo.pdf [Última vez visitado, 22/12/1999]. November, 1999.

Riehle, Dirk. "*Working with Classes and Interfaces*". C++ Report, 12(3):14-21. March, 2000.

Rising, Linda. "*Design Patterns: Elements of Reusable Architectures*". Annual Review of Communications, Vol. 49:907-909. 1996.

Rojas, Alfonso y Lozano, Javier. "*Excepciones en C++ para DOS. La Última Herramienta para el ANSI C++*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(6):43-47. Abril, 1995.

Rumbaugh, James E. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). ACM. Reprinted in ACM SIGPLAN 22(12):466-481. October, 1987.

Rumbaugh, James. "*Depending on Collaborations: Dependencies as Contextual Associations*". Journal of Object-Oriented Programming (JOOP), 11(4):5-9. July/August, 1998.

Schimidt, Douglas C. "*Introduction to Design Patterns*". Washington University, St. Louis. http://www.cs.wustl.edu/~schmidt/cs242/patterns-intro4.ps.gz [Última vez visitado, 13-3-2000]. 1998.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". STL Documentation. October. 1995.

Stroustrup, Bjarne. "*The Design and Evolution of C++*". Addison-Wesley, 1994. Reprinted with corrections in April, 1995.

Wegner, Peter. "*Dimensions of Object-Oriented Modeling*". IEEE Computer, 25(10):12-20. October, 1992.

Bibliografía para la preparación de las clases teóricas

"*Proceedings of the 3rd Pattern Languages of Programming Conference- PLoP'96*". Allerton Park, Illinois, Sept. 4-6, 1996. Washington University Technical Report (#wucs-97-07). Available online at http://www.cs.wustl.edu/~schmidt/PLoP-96/workshops.html. [Última vez visitado, 13-3-2000]. 1996.

"*Proceedings of the 1stEuropean Conference on Pattern Languages of Programming - EuroPLoP '96*". Kloster Irsee, Germany, July 11-13, 1996. Washington University Technical Report (#wucs-97-07). Available online at http://www.cs.wustl.edu/~schmidt/europlop-96/writers-workshop.html. [Última vez visitado, 13-3-2000]. 1996.

"*Proceedings of the 4th Pattern Languages of Programming Conference- PLoP'97*". September 3-5, 1997 Allerton Park Monticello, Illinois, USA. Washington University Technical Report wucs-97-34. Available online at http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/proceedings.zip. [Última vez visitado, 13-3-2000]. 1997.

"*Proceedings of the 2nd European Conference on Pattern Languages of Programming - EuroPLoP '97*". Siemens Technical Report 120/SW1/FB. Munich, Germany. Available online at http://www.riehle.org/events/europlop-1997/index.html [Última vez visitado, 13-3-2000]. 1997.

"*Proceedings of the 1998 Pattern Languages of Programs Conference – PLoP'98*". August 11-14, 1998 Allerton Park Monticello, Illinois, USA. Washington University Technical Report TR #WUCS-98-25. Available online at http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/ [Última vez visitado, 13-3-2000]. 1998.

"*Proceedings of the 1999 Pattern Languages of Programs Conference – PLoP'99*". August 1999. Allerton Park Monticello, Illinois, USA. Available online at http://st-www.cs.uiuc.edu/~plop/plop99/proceedings/. [Última vez visitado, 13-3-2000]. 1999.

"*Proceedings of the Fourth European Conference on Pattern Languages of Programming and Computing – EuroPLoP'99*". 8 - 10 July 1999. Bad Irsee, Germany. Available online at http://www.argo.be/europlop/writers.htm. [Última vez visitado, 13-3-2000]. 1999.

Adams, Rolf. "*Visitor or Strategy? A Comparison*". Journal of Object-Oriented Programming (JOOP), 13(1):27-28. March/April, 2000.

Aklecha, Vishwajit. "*Object-Oriented Frameworks Using C++ and CORBA Gold Book*". Coriolis Technology Press, 1999.

Al-Ahmad, W. and Steegmans, E. "*Inheritance in Object-Oriented Languages: Requirements and Supporting Mechanisms*". Journal of Object-Oriented Programming (JOOP), 12(8): 15-24,48. January, 2000.

Anaya de Paez, Raquel. "*Desarrollo de Componentes Reutilizables en el Marco de OASIS*". Tesis Doctoral. Universidad Politécnica de Valencia. 1999.

Arnold, Ken and Gosling, James. "*El Lenguaje de Programación Java*". Addison-Wesley/Domo, 1997.

Bäumer, Dirk, Gryczan, Guido, Knoll, Rolf, Lilienthal, Carola, Riehle, Dirk and Züllighoven, Heinz. "*Framework Development for Large Systems*". Communications of the ACM, 40(10):52-59. October, 1997.

Beck, Kent. "*Patterns and Software Development. Adding Value to Reusable Software*". Dr. Dobb's Journal on CD-ROM. February, 1994.

Beck, Kent, Coplien, James O., Crocker, Ron, Dominick, Lutz, Meszaros, Gerard, Paulisch, Frances and Vlissides, John. "*Industrial Experience with Design Patterns*". In Proceedings of the 18th International Conference on Software Engineering - ICSE '96. (March 25-29, 1996, Berlin, Germany). Pages 103-114. ACM, 1996.

Berard, Edward V. "*Object-Oriented Programming Languages*". The Object Agency. http://www.toa.com/pub/oopl_article.txt. [Última vez visitado, 24-2-2000]. 1989.

Booch, Grady. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Budd, Timothy. "*Programación Orientada a Objetos*". Addison-Wesley Iberoamericana, 1994.

Budinsky, M. A., Finnie, M. A., Vlissides, J. M. and Yu, P. S. "*Automatic Code Generation from Design Patterns*". IBM Systems Journal, 35(2). 1996. Also online avaliable in http://www.reasearch.ibm.com/journal/sj/budin/budinsky.html. [Última vez visitado, 12-3-2000].

Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad, Peter and Stal, Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Carlson, Brent. "*Design Patterns for Business Frameworks and Applications*". Java Report, 5(3):34-46. March, 2000.

Cleeland, Chris and Schmidt, Douglas C. "*External Polymorphism. An Object Structural Pattern for Transparently Extending Concrete Data Types*". In *Patterns Languages of Program Design 3*. Martin, R., Riehle, D. and Buschmann, F. editors. Pages 377-390. Addison-Wesley, 1997.

Coad, Peter. "*Finding Objects: Practical Approaches*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'91. (October 6 - 11, 1991, Phoenix, AZ USA). Pages 17-19. ACM, 1991.

Coad, Peter, North, David and Mayfield, Mark. "*Object Models. Strategies, Patterns, & Applications*". 2nd Edition. Yourdon Press Computing Series. Prentice Hall, 1997.

Cockburn, Alistair. "*The Interaction of Social Issues and Software Architecture*". Communications of the ACM, 39(10):40-46. October, 1996.

Codenie, Wim, Hondt, Koen De, Steyaert, Patrick and Vercammen, Arlette. "*From Custom Applications to Domain-Specific Frameworks*". Communications of the ACM, 40(10):71-77. October, 1997.

Coldewey, J. and Dyson, P. (editors). "*Proceedings of the Third European Conference on Pattern Languages of Programming and Computing – EuroPLoP'98*". 9 - 11 July 1998 Kloster Irsee, Germany. Available online at http://www.coldewey.com/europlop98/Program/writers.htm [Última vez visitado, 13-3-2000]. Universitätsverlag Konstanz Technicl Report. July, 1998.

Coplien, James O. "*Idioms and Patterns as Architectural Literature*". IEEE Software, 14(1):36-42. January, 1997.

Coplien, James O. "*Space: The Final Frontier*". C++ Report, 10(3):11-17. March, 1998.

Coplien, James O. "*Baa Baa Baaa*". C++ Report, 11(9):33-37. October, 1999.

Coplien, James O. and Schmidt, Douglas C. (editors) "*Patterns Languages of Program Design*". Addison-Wesley, 1995.

Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

Crespo, Yania, Marqués, Corral y Rodríguez, Juan José. "*Genericidad Inversa*". En las Actas de las II Jornadas de Trabajo MENHIR. Editor J. Á. Carsí. (19-20 de febrero de 1998. Valencia - España). Páginas 143-148. Organizado por el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. Febrero, 1998.

Crespo, Yania, Marqués, Corral y Rodríguez, Juan José. "*Transformación de Clases para Obtener Clases Genéricas. Implicaciones en las Jerarquías de Herencia y de Agregación/Composición*". Technical Report TR-GIRO-04-98. Universidad de Valladolid. 1998.

Crespo, Yania, Rodríguez, Juan José, García, Francisco José y Marqués, José Manuel. "*Obtención Automática de Clases Genéricas a través de una Operación de Parametrización*". En Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISDB'99. Editores Pere Botella, Juan Hernández y Félix Saltor. (Cáceres, 24-26 de Noviembre de 1999):343-354. 1999.

Cybulski, Jacob L. "*Introduction to Software Reuse*". Technical Report TR 96/4, Department of Information Systems. University of Melbourne (Australia). July, 1996.

Champeaux, Dennis de, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison-Wesley, 1993. HTML Edition available at http://gee.cd.oswego.edu/dl/oosdw3. [Última vez visitado, 1-2-2000].

Deadman, Richard. "*When in Rome: A Guide to the Java Paradigm*". Java Report, 2(9):41-52,70. October, 1997.

Demeyer, Serge. "*Zypher: Tailorability as a Link from Object-Oriented Software Engineering to Open Hypermedia*". Phd dissertation. Faculty of Sciences, Brussels Free University - July, 1996. http://dinf.vub.ac.be/~demeyer/Zypher/zpprpdf/ [Última vez visitado, 13-3-2000]. Last revision: January, 29th 1997.

Demeyer, Serge, Meijler, Theo Dirk, Nierstrasz and Steyaert, Patrick. "*Design Guidelines for Tailorable Frameworks*". Communications of the ACM, 40(10):60-64. October, 1997.

Devis Botella, Ricardo. "*Smalltalk: Una Aproximación Práctica*". Revista Profesional para Programadores (RPP), Editorial Anaya Multimedia, II(6):16-24. Abril, 1995.

Devis Botella, Ricardo. "*C++. STL, Plantillas, Excepciones, Roles y Objetos*". Paraninfo, 1997.

Donadi, Mahesh. "*Rules Are for Fools, Patterns Are for Cools Fools*". Journal of Object-Oriented Programming (JOOP), 12(6):21-23,70. October, 1999.

Dodani, Mahesh. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January, 1999.

Duell, Michael. "*Non-Software Examples of Software Design Patterns*". Object Magazine, 7(5):52-57. July, 1997.

Durán Toro, Amador. "*Apuntes de Ingeniería del software II*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/publicaciones.html [Última vez visitado, 14-3-2000]. 2000.

Fayad, Mohamed E. and Schmidt, Douglas C. (guest editors) "*Object-Oriented Application Frameworks*". Communications of the ACM, 40(10):32-38. October, 1997.

Foster, Ted and Zhao, Liping. "*Cascade*". Journal of Object-Oriented Programming (JOOP), 11(9):18-24. February, 1999.

Fowler, Martin. "*Analysis Patterns. Reusable Object Models*". Object Technology Series. Addison-Wesley, 1997.

Gabrilovich, Evgeniy. "*Destruction-Managed Singleton. A Compound Pattern for Reliable Deallocation of Singles*". C++ Report, 12(3):35-40,47. March, 2000.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre, 1997.

García Peñalvo, Francisco José y Crespo González-Carvajal, Yania. "*Las Implicaciones de Eiffel en el Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(7):45-53. Julio, 1998.

Garlow, Joanne, Holmes, Chris and Mowbray, Thomas. "*Applying Design Patterns in UML*". Rose Architect Magazine, 1(2). Summer, 1999. http://www.rosearchitect.com/mag/archives/9901/f3.stml. [Última vez visitado, 15-2-2000].

Gil, Joseph and Lorenz, David H. "*Design Patterns and Language Design*". IEEE Computer, 31(3):118-129. March, 1998.

Girardi, M. Rosario. "*Main Approaches to Software Classification and Retrieval*". En las actas del curso Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

Goldfedder, Brandon and Rising, Linda. "*A Training Experience with Patterns*". Communications of ACM, 39(10):60-64. October, 1996.

Graham, Ian. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Grand, Mark. "*Paterns in Java Volume 1*". John Wiley & Sons, 1998.

Grand, Mark. "*Paterns in Java Volume 2"*. John Wiley & Sons, 1999

Harrison, Neil, Foote, Brian and Rohnert, Hans (editors). "*Patterns Languages of Program Design 4*". Addison-Wesley, 2000.

Helm, Richard and Gamma, Erich. "*Pattern and Software Design. Designing Objects for Extension*". Dr. Dobb's on CD-ROM. September, 1995.

Henney, Kevlin. "*Somethig for Nothing*". Java Report, 4(12):74-80. December, 1999.

Hu, Weizhen. "*Singleton Pattern: Handy, but Be Careful*". Java Report, 5(3):28-32, 76. March, 2000.

Jacobson, Ivar, Griss, Martín L. and Jonsson, Patrik. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley, 1997.

Johnson, Ralph E. "*Documenting Frameworks Using Patterns*". In Conference Proceedings on Object-oriented Programming Systems, Languages, and Applications – OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 63-76. ACM, 1992.

Johnson, Ralph E. "*Frameworks = (Components + Patterns)*". Communications of the ACM, 40(10):39-42. October, 1997.

Johnson, Ralph E. and Foote, Brian. "*Designing Reusable Classes*". Journal of Object-Oriented Programming, 1(2):22-35. 1988.

Johnson, Ralph E. and Russo, Vincent F. "*Reusing Object-Oriented Designs*". Technical Report UIUCDCS 91-1696, University of Illinois, 1991.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

Katrib Mora, Miguel. "*C++ Versus Eiffel (I)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(10):73-78. Noviembre, 1996.

Katrib Mora, Miguel. "*C++ Versus Eiffel (II)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, III(11):69-76. Diciembre, 1996.

Katrib Mora, Miguel. "*C++ Versus Eiffel (y III)*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, IV(1):67-74. Enero, 1997.

Kerth, Norman L. and Cunningham, Ward. "*Using Patterns to Improve Our Architectural Vision*". IEEE Software, 14(1):53-59. January, 1997.

Kinoshita, Shigeyuki (leader), Yoshikawa, Hiroshi (sub-leader), Matsumoto, Yoshihiro, Nakase, Masaharu, Mashiyama, Yohei, Tsunekawa, Ikuyo, Miki, Ryoji, Yakazi,

Tomoo, Furukawa, Yohichiro and Ishikawa, Yuichi. "*State of the Art of Reuse in Object-Oriented Development*". Japan GUIDE/SHARE. http://www.guide.org/jgs/jgsoo1.htm. June, 1996.

Kühne, Thomas. "*The Function Object Pattern*". C++ Report, 9(9):32-42. October, 1997**.**

Kurotsuchi, Brian T. "*Welcome to the wonderful world of Design Patterns*". http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/. [Última vez visitado, 15-3-2000]. 1996.

Larman, Craig. "*The Presentation and Domain Layers*". Java Report, 4(12):64-68,82. December, 1999.

Larman, Craig. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Larman, Craig. "*Designing Object Responsibilities and Collaborations*". Java Report, 5(3):69-75. March, 2000.

Larsen, Grant. "*Designing Component-Based Frameworks Using Patterns in the UML*". Communications of the ACM, 42(10):38-45. October, 1999.

Larsson, Johan. "*An Introduction to Customization Design Patterns in EFLIB*". Journal of Object-Oriented Programming (JOOP), 12(5):24-28. September, 1999.

Lea, Doug. "*Christopher Alexander: An Introduction for Object-Oriented Designers*". ACM Software Engineering Notes, 19(1):39-46. January, 1994.

Lea, Doug. "*Design Patterns for Avionics Control Systems*". DRAFT Version 0.9.6. Technical Report DSSA Adage Project ADAGE-OSW-94-01. http://gee.cs.oswego.edu/dl/acs/acs.pdf. [Última vez visitado, 15-3-2000]. November 20, 1994.

Mannion, Mike. "*Dynamic Binder*". Java Report, 5(3):20-26. March, 2000.

Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

Marqués Corral, José Manuel. "*Reutilización Sistemática del Software*". Notas de conferencia. Escuela Universitaria de Ingeniería Técnica en Informática de Gestión – Edificio Politécnico – Ourense, 27 de noviembre de 1998.

Marqués Corral, José Manuel. "*Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación*". En las actas del curso *Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática*. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo – Ourense, del 6 al 10 de julio de 1998). Julio, 1998.

Martin, Robert C. "*The Open Closed Principle*". C++ Report, 8(1). January, 1996.

Martin, Robert C. "*The Liskov Substitution Principle*". C++ Report, 8(3). March, 1996.

Martin, Robert C. "*The Dependency Inversion Principle*". C++ Report, 8(5). May, 1996.

Martin, Robert C. "*The Interface Segregation Principle*". C++ Report, 8(7). July-August, 1996.

Martin, Robert C. "*Granularity*". C++ Report, 8(10). November-December, 1996.

Martin, Robert C. "*Stability*". C++ Report, 9(2). February, 1997.

Martin, Robert, Riehle, Dirk and Buschmann, Frank (editors). "*Patterns Languages of Program Design 3*". Addison-Wesley, 1998.

McGregor, John D. "*Test Patterns: Please Stand By*". Journal of Object-Oriented Programming (JOOP), 12(3):14-18. June, 1999.

Meszaros, Gerard and Doble, Jim. "*A Pattern Language for Pattern Writing*". http://hillside.net/patterns/Writing/pattern_index.html. [Última vez visitado, 16-3-2000]. 1996.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Monroe, Robert T., Kompanek, Andrew, Melton, Ralph and Garlan, David. "*Architectural Styles, Design Patterns, and Objects*". IEEE Software, 14(1):43-52. January, 1997.

Moros Valle, Begoña, Nicolás Ros, Joaquín, García Molina, Jesús and Toval Álvarez, José Ambrosio. "*Combining Formal Specifications with Design by Contract*". Journal of Object-Oriented Programming, 12(9):16-21,46. February, 2000.

Nierstrasz, Oscar, Gibbs, Simon and Tsichritzis, Dennis. "*Component-Oriented Software Development*". Communications of the ACM, 33(9):160-165. September, 1992.

Norris, Eugene N. "*Lecture Notes of the CS 699-001 – Object-Oriented Design and Implementation in C++ and Java*". http://www.cs.gmu.edu/~Eenorris/O-Ocourses/699Syllabus.html. Spring 1998.

Oaks, Scott. "*Creating Good Java Interfaces*". Java Report, 4(12):88,86. December, 1999.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Pescio, Carlo. "*Principles Versus Patterns*". IEEE Computer, 30(9):130-131. September, 1997.

Pescio, Carlo. "*Deriving Patterns from Design Principles*". Journal of Object-Oriented Programming (JOOP), 11(6):67-71. October, 1998.

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Prechelt, Lutz, Unger, Barbara and Schmidt, Douglas C. "*Replication of the First Controlled Experiment on the Usefulness of Design Patterns: Detailed Description and Evaluation*". Technical Report wucs-97-34. Department of Computer Science. Washington University, St. Louis, MO 63130-4899. http://www.cs.wustl.edu/cs/techreports/1997. December, 1997.

Preiss, Bruno R. "*Data Structures and Algorithms with Object-Oriented Design Patterns in C++*". John Wiley & Sons, 1999. Web Book version available at http://www.pads.uwaterloo.ca/Bruno.Preiss/books/opus4/html/book.html. [Última vez visitado, 16-3-2000].

Preiss, Bruno R. "*Data Structures and Algorithms with Object-Oriented Design Patterns in Java*". John Wiley & Sons, 2000. Web Book version available at http://www.pads.uwaterloo.ca/Bruno.Preiss/books/opus5/. [Última vez visitado, 16-3-2000].

Prieto Arambillet, Félix. "*Apuntes de la Asignatura Programación III*". Versión 1.0. Departamento de Informática. Universidad de Valladolid. Diciembre, 1999.

Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

Reeves, Jack W. "*More on Template Specialization*". C++ Report, 12(2):16-20. February, 2000.

Rensselaer Polytechnic Institute. "*Standard Template Library Reference*". Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/~musser/stl. [Última vez visitado, 6-3-2000]. 1994.

Richards, Ben H. "*Frameworks and Design Patterns*". MultiUse Express, 5(1):2. February, 1997.

Rising, Linda. "*The Road, Christopher Alexander, and Good Software Design*". Object Magazine. Pages 47-50. March, 1997.

Rising, Linda. "*The Patterns Almanac 2000*". Addison-Wesley, 2000.

Rodríguez, Juan José, Crespo, Yania y Marqués, Corral. "*Transformación de Jerarquías de Herencia Múltiple en Jerarquías de Herencia Sencilla*". Technical Report TR-GIRO-03-98. Universidad de Valladolid. 1998.

Rumbaugh, James. "*Controlling Propagation of Operations Using Attributes on Relations*". In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications – OOPSLA'88. (September 25 - 30, 1988, San Diego, CA USA). Pages 285-296. ACM, 1988.

Rumbaugh, James, Blaha, **Michael, Premerlani, William, Eddy, Frederick, Lorensen, William**. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Schmid, Hans Albrecht. "*Systematic Framework Design by Generalization*". Communications of the ACM, 40(10):48-51. October, 1997.

Schimidt, Douglas C. "*Using Design Patterns to Develop Reusable Object-Oriented Communication Software*". Communications of the ACM, 38(10). October, 1995.

Schimidt, Douglas C. "*Advanced C++ Features, Design Patterns, and Frameworks*". Washington University, St. Louis. http://www.cs.wustl.edu/~schmidt/cs242/pattern-examples4.ps.gz [Última vez visitado, 13-3-2000]. 1998.

Schmidt, Douglas C. and Fayad, Mohamed E.. "*Lessons Learned. Building Reusable OO Frameworks for Distributed Software*". Communications of the ACM, 40(10):85-87. October, 1997.

Schmidt, Douglas C. and Stephenson, Paul. "*Using Design Patterns to Evolve System Software from UNIX to Windows NT*". C++ Report. March/April, 1995.

Schmidt, Douglas C. and Stephenson, Paul. "*Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms*". In Proceedings of the 9th European Conference on Object-Oriented Programming – ECOOP'95. (Aarhus, Denmark on August 7-11, 1995). 1995.

Schmidt, Douglas C., Fayad, Mohamed and Johnson, Ralph E. (guest editors). "*Software Patterns*". Communications of the ACM, 39(10):36-39. October, 1996.

Schmidt, Douglas C., Stal, Michael and Buschmann, Frank. "*Patterns for Concurrent and Distributed Objects – POSA2*". To be published by John Wiley & Sons in 2000. Many drafts are available at http://www.cs.wustl.edu/~schmidt/patterns/patterns.html [Última vez visitado, 13-3-2000]. 2000.

Shaw, Mary and Garlan, David. "*Software Architecture. Perspectives on an Emerging Discipline*". Prentice Hall, 1996.

Smith, Mark L. "*An STL-Based N-Way Set*". C/C++ Users Journal, 18(3):76-83. March, 2000.

Stroustrup, Bjarne. "*El Lenguaje de Programación C++*". 3ª Edición. Addison-Wesley, 1998.

Sutter, Herb. "*Standard Library News, Part 1: Vectors and Deques*". C++ Report, 11(7). July/August, 1999.

Sutter, Herb. "*Standard Library News: Sets and Maps*". C++ Report, 11(9):38-41. October, 1999.

Taligent, Inc. "*Leveraging Object-Oriented Frameworks*". Taligent White Papers. IBM. http://www.ibm.com/java/education/ooleveraging/index.html [Última vez visitado, 13-3-2000]. 1993.

Taligent, Inc. "*Building Object-Oriented Frameworks*". Taligent White Papers. IBM. http://www.ibm.com/java/education/oobuilding/index.html [Última vez visitado, 13-3-2000]. 1994.

Tepfenhart, William M. and Cusick, James J. "*A Unified Object Topology*". IEEE Software, 14(1):31-35. January, 1997.

Vlissides, John. "*Pattern Hatching. Design Patterns Applied*". Software Patterns Series. Addison-Wesley, 1998.

Vlissides, John and Alexandrescu, Andrei. "*To Code or Not to Code, Part 1*". C++ Report, 12(3):44-47. March, 2000.

Vlissides, J., Coplien, J. and Kerth, N. (editors) "*Patterns Languages of Programming Design 2*". Addison-Wesley, 1996.

Weidl, Johannes. "*The Standard Template Library Tutorial*". Technical Report 184.437 Wahlfachpraktikum (10.0). Information Systems Institute. Distributed Systems Department. Technical University Vienna. http://www.infosys.tuwien.ac.at/Research/Component/Tuturial/prwmain.htm. [Última vez visitado, 6-3-2000]. April, 1996.

Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

Wirfs-Brock, Allen, Vlissides, John, Cunningham, Ward, Johnson, Ralph and Bollette, Lonnie. "*Panel: Designing Reusable Designs: Experiences Designing Object-Oriented Frameworks*". In Addendum to the Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA90/ECOOP90. (October 21 - 25, 1990, Ottawa Canada). Pages 19-24. ACM, 1990.

Yacoub, Sherif M. and Ammar, H. H. "*Towards Pattern-Oriented Frameworks*". Journal of Object-Oriented Programming (JOOP), 12(8):25-38,46. January, 2000.

5.3.2 Programa de la parte práctica

La parte práctica de la asignatura de Programación Orientada a Objetos está dirigida a satisfacer aquellos objetivos prácticos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**P2** y **P5**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000] (*incluido en el Apéndice A de este Proyecto Docente*).

P2	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
P5	Programación orientada a objetos.

Tabla 5.32. Objetivos del programa de prácticas de la asignatura Programación Orientada a Objetos

5.3.2.1 Consideraciones iniciales

El objetivo fundamental de las prácticas de la asignatura de Programación Orientada a Objetos es el diseño de sistemas software bajo el paradigma objetual, así como su posterior implementación en un lenguaje de programación orientado a objetos como puede ser C++.

Se considera que las 30 horas, que equivalen a los tres créditos de la parte práctica, deben repartirse en prácticas presenciales, transmisión de contenidos en laboratorio y algún taller, así como prácticas libres, que los alumnos deben aprovechar para entrenarse con el lenguaje de programación elegido, C++, y realizar la práctica obligatoria.

Para el mejor aprovechamiento de las clases presenciales, éstas se llevan a cabo una vez cada semana durante dos horas consecutivas.

5.3.2.2 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se ha organizado en tres bloques prácticos: *laboratorio (que combina la transmisión de conocimientos con las prácticas libres), taller y prácticas obligatoria*, como se muestra en la Tabla 5.33.

En la Tabla 5.34 se presenta la correspondencia existente entre el programa de la parte práctica y los objetivos prácticos perseguidos en esta asignatura.

Laboratorio (18 Horas)

Práctica 1. Modelo objeto en C++ (10 Horas)

Práctica 2. STL (8 Horas)

Taller (2 Horas)

Práctica 3. Tarjetas CRC

Práctica obligatoria (Prácticas libres)

Práctica 4. Diseño e implementación de una aplicación en C++

Tabla 5.33. Estructura del programa de prácticas de la asignatura Programación Orientada a Objetos (3 créditos)

Elemento Docente	Objetivos
Práctica 1	P5
Práctica 2	P5
Práctica 3	P2, H1, H2, H3, H4
Práctica 4	P2, P5, H1, H2, H3, H4

Tabla 5.34. Correspondencia entre el temario de prácticas y los objetivos prácticos de la asignatura

Desarrollo de las clases prácticas (laboratorio y taller)

De los tres bloques en los que se ha dividido la parte práctica, dos conllevan la presencia de los alumnos, a saber: las clases en el laboratorio y el taller.

Las clases en el laboratorio combinan la transmisión conocimientos con prácticas libres. Su cometido es completar los apartados del programa de teoría que hacen referencia a C++ y a la biblioteca estándar de plantillas (STL).

Se ha intentado minimizar el tiempo de exposición por parte del profesor, para que los alumnos tengan el mayor tiempo posible para aplicar en la práctica los conceptos transmitidos. Este planteamiento se justifica en los siguientes puntos:

- El énfasis de la asignatura no debe hacerse en la sintaxis del lenguaje elegido, máxime cuando son alumnos de tercer curso de una Ingeniería Informática.
- Se ha deseado romper, en cierto grado, la excesiva vinculación de los alumnos con respecto al profesor. Se busca que antes de que busquen al profesor para resolver una duda en su trabajo práctico, intenten solventarla por ellos mismos.

El taller se lleva a cabo en la clase de teoría. Los alumnos se agrupan en grupos de entre cuatro y seis personas que durante aproximadamente una hora de las dos de que se disponen, discuten y desarrollan la solución a un problema cuyo enunciado, previamente, ha sido facilitado a través de la página de la asignatura.

Tras su exposición, se desarrolla un debate con el resto de los grupos, moderado por el profesor. De esta manera se logra, por una parte, que los alumnos se expresen en público y defiendan su trabajo, y por otra la corrección pública de los errores cometidos en los modelos.

Con los comentarios, el grupo encargado de la defensa elabora un informe que entregan al profesor, quien, tras comprobar que no contiene errores, lo publicará en la página de la asignatura para que sirva de material de estudio al resto de los alumnos, y para promociones futuras.

El tema elegido para el taller es la identificación de clases utilizando tarjetas CRC.

Evaluación de la parte práctica

La forma principal de evaluar la parte práctica de esta asignatura es mediante la realización de una práctica obligatoria, cuyos requisitos deben ser fáciles de obtener, utilizándose como tema del trabajo algo sobradamente conocido por los alumnos, como un juego o algún aspecto relacionado con sus proyectos de final de carrera.

En la Figura 5.11 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota de la práctica obligatoria y los informes de los talleres, realizados voluntariamente.

Si (Teoría ≥ 4,75) **y** (***Práctica*** ≥ 5.0)
    Nota Final = (Teoría*0,5) + (***Práctica****0,5) + ***Nota trabajos***
Sino
    Ø
Fin si

Figura 5.11. Influencia de la nota en la parte práctica en la nota final de Programación Orientada a Objetos

Bibliografía básica de referencia

De la lista de títulos recomendados, los siguientes pueden ser los más adecuados para ser consultados en al realización de la parte práctica.

- **Deitel, H. M. and Deitel, P. J.** "*C++ How To Program*". 2nd Edition. Prentice Hall, 1998.
- **Glass, Graham and Schuchert, Brett**. "*The STL <Primer>*". Prentice Hall, 1996.
- **OMG.** "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. [Última vez visitado, 14/2/2000]. June, 1999.
- **Stroustrup, Bjarne.** "*El Lenguaje de Programación C++*". 3ª Edición Addison-Wesley, 1998.

5.3.2.3 Desarrollo comentado del programa

En el programa de práctica se distinguen tres bloques. El primero está compuesto por las clases en el laboratorio, donde se va a hacer hincapié en los aspectos relacionados con el lenguaje de programación elegido. El segundo sería un taller, donde se realizará un supuesto práctico utilizando tarjetas de clase. El tercero, y último, es el que se ocupa de la práctica obligatoria, por lo tanto práctica no presencial.

Laboratorio

El laboratorio juega un doble papel en la asignatura. Por un lado al docente le permite completar los contenidos teóricos, con aquellos aspectos relacionados con el lenguaje de programación elegido, C++ en este caso. Por otro lado, el alumno lleva a la práctica los conocimientos adquiridos.

Los objetivos generales de estas prácticas se pueden resumir en los siguientes puntos:

- Presentar el modelo objeto de C++.
- Introducción a la STL.
- Dominio práctico de los temas anteriores por parte del alumno.

Las clases en el laboratorio quedan divididas en dos temas prácticos: *el modelo objeto de C++* y *la biblioteca estándar de plantilla de C++ (STL)*.

Este bloque práctico se lleva el **60%** del tiempo dedicado a las prácticas.

Práctica 1. El Modelo Objeto de C++

Busca introducir al alumno en los conceptos orientados al objeto de este lenguaje de programación híbrido, partiendo del hecho de que todos los matriculados dominan el lenguaje C.

El esquema de temas a abordar en este apartado es el siguiente:

a. Introducción
b. Recursos básicos
c. Clases
d. Sobrecarga de operadores
e. Herencia
f. Polimorfismo
g. Plantillas
h. Excepciones

Las dos referencias que se utilizan mayormente en esta parte son:

- **Deitel, H. M. and Deitel, P. J.** "*C++ How To Program*". 2nd Edition. Prentice Hall, 1998.
- **Stroustrup, Bjarne**. "*El Lenguaje de Programación C++*". 3ª Edición Addison-Wesley, 1998.

Práctica 2. STL

Introduce la biblioteca estándar de plantillas de C++, STL. Esta biblioteca ofrece un conjunto de componentes bien estructurados que trabajan de una forma adecuada juntos. Está diseñada de forma que se integra perfectamente con código C++.

El esquema de temas a tratar es el siguiente:

a. ***Introducción***
b. ***Contenedores***
c. ***Iteradores***
d. ***Algoritmos***

Las referencias más utilizadas para esta parte son:

- **Glass, Graham and Schuchert, Brett**. "*The STL <Primer>*". Prentice Hall, 1996.
- **Kirman, Jak.** "*A Modest STL Tutorial*". http://www.cs.brown.edu/people/jak/proglan/stltut/tut.html. [Última vez visitado, 10-11-1999]. January 1998.
- **Rensselaer Polytechnic Institute.** "*Standard Template Library Reference*". Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/~musser/stl. [Última vez visitado, 6-3-2000]. 1994.
- **Stepanov, A. A. and Lee, M.** "*The Standard Template Library*". STL Documentation. October. 1995.
- **Weidl, Johannes.** "*The Standard Template Library Tutorial*". Technical Report 184.437 Wahlfachpraktikum (10.0). Information Systems Institute. Distributed Systems Department. Technical University Vienna. http://www.infosys.tuwien.ac.at/Research/Component/Tuturial/prwmain.htm. [Última vez visitado, 6-3-2000]. April, 1996.

Taller

Los talleres no son el punto central de las prácticas de esta asignatura, como ocurría con Ingeniería del Software, pero si pueden servir para discutir algún aspecto de modelado o de diseño en la asignatura.

En concreto, se ha reservado aproximadamente el **6,5%** de la parte práctica a talleres, esto es una sesión de dos horas, donde el tema a tratar es la identificación de las clases semánticas en un supuesto práctico, utilizándose para ello las tarjetas de clases o tarjetas CRC (*Classes, Responsabilities and Collaborations*) [Beck and Cunningham, 1989].

La justificación de este taller se basa en que los alumnos, acostumbrados a afrontar los desarrollos con una mentalidad puramente estructurada, tienen problemas a la hora

de identificar los objetos semánticos que van a dar el soporte a la aplicación a desarrollar. Por este motivo, para la realización de este taller se utiliza una técnica sumamente intuitiva como son las tarjetas de clase.

Práctica obligatoria

Se convierte en el elemento fundamental para la evaluación de la parte práctica de la asignatura de Programación Orientada a Objetos.

En la presentación de la asignatura se les presenta a los alumnos la necesidad de realizar por parejas el desarrollo de una aplicación orientada a objetos en C++ para superar la parte práctica. El tema de la misma debe ser conocido por los alumnos para no perder tiempo en la captura de requisitos, recomendándose la realización de un juego de mesa o de una parte relacionada con sus proyectos de final de carrera.

La práctica así planteada presenta los siguientes aspectos positivos:

- Se sigue una política de trabajo en grupo. Además, como el grupo es reducido es más difícil que un integrante no participe en la elaboración de la práctica.
- Cada trabajo realizado es diferente, con lo que se evita el plagio de prácticas.
- Supone un tema atractivo y no excesivamente complejo

Los productos a entregar son:

- *Un documento de* ***especificación de diseño***. Que además de una introducción a la aplicación que se desarrolla, presente el diseño realizado en UML de la misma.
- *Manual del programador*. Describiendo las bibliotecas de clases implementadas (debe incluir una guía para la compilación de los fuentes).
- *Fuentes.*
- *Ejecutable.*

Cada grupo deberá entregar todos los ficheros de su trabajo organizados en documentación y ejecutables, así como la especificación de requisitos impresa.

Después de la entrega se publicará un calendario para la defensa por grupos de la práctica.

La defensa de la práctica se realizará en un examen oral, donde durante aproximadamente unos 15 minutos los integrantes del grupo contestarán a preguntas sobre las características técnicas de la aplicación realizada.

Al ser un trabajo realizado en grupo, todos los integrantes del mismo recibirán la misma nota. Esto significa que la actuación individual de cada integrante repercutirá en el global del grupo.

Con el fin de promover una mayor motivación hacia el trabajo, y por transitividad hacia la asignatura, la nota final del trabajo dependerá de un baremo impuesto en función de la calidad técnica y de la presentación de los trabajos. De todas formas, con independencia de la nota final, para que la práctica se considere superada se hará uso de la siguiente fórmula:

$$(NotaTécnica*0,8)+(PresentaciónOral*0,1)+(PresentaciónDocumentación*0,1) \geq 5$$

En general, con esta práctica se busca satisfacer en general todos los objetivos prácticos identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

De forma más concreta se podían citar los siguientes objetivos específicos:

- *Que los alumnos se enfrenten al diseño e implementación de una aplicación bajo el paradigma orientado a objetos.*
- *Que los alumnos entiendan la diferencia entre un programa realizado con técnicas estructuras y otro realizado orientado al objeto.*
- *Que los alumnos pongan en práctica las técnicas explicadas en la parte teórica y en los talleres de la asignatura.*
- *Que los alumnos participen en un trabajo en grupo.*
- *Que los alumnos se acostumbren a documentar el software realizando, cogiendo soltura en la redacción de documentos técnicos.*
- *Que los alumnos manejen bibliografía especializada para profundizar en los aspectos teóricos que dan soporte a la práctica.*
- *Que los alumnos desarrollen y mejoren su capacidad de comunicación.*

5.3.2.4 Recursos para desarrollar la parte práctica

En el presente apartado se enumeran diferentes recursos bibliográficos y software que se pueden aprovechar para impartir la asignatura, tanto este año como en años venideros, teniendo en cuenta la posible utilización de otros lenguajes de programación diferentes a C++.

Bibliografía de C++

Alonso Romero, Luis, Cabana González, Juan José, Castro Odio, Guido y Crespo González-Carvajal, Yania. "*Programación Avanzada con Visual C++*". Documentación del Curso Extraordinario de la Universidad de Salamanca. Departamento de Informática y Automática. Facultad de Ciencias. Universidad de Salamanca. Noviembre, 1999.

Alonso Amo, F. y Segovia Pérez, F. J. "*Entornos y Metodologías de Programación*". Paraninfo, 1995.

Andrews, Mark. "*Aprenda Visual C++ Ya*". McGraw-Hill, 1997.

Brokken, Frank B. "*C++ Annotations Version 4.4.1c*". ICCE, University of Groningen Grote Rozenstraat 38, 9712 TJ Groningen Netherlands. Published at the University of Groningen ISBN 90 367 0470 7. Available at http://www.icce.rug.nl/docs/cplusplus/cplusplus.html [Última vez visitado, 13-3-200]. 2000.

Deitel, H. M. and Deitel, P. J. "*Cómo Programar en C/C++*". 2ª Edición. Prentice Hall Hispanoamericana, 1995.

Deitel, H. M. and Deitel, P. J. "*C++ How To Program*". 2nd Edition. Prentice Hall, 1998.

Devis Botella, Ricardo. "*Programación Orientada a Objetos en C++*". Paraninfo, 1993.

Devis Botella, Ricardo. "*C++. STL, Plantillas, Excepciones, Roles y Objetos*". Paraninfo, 1997.

Eckel, Bruce. "*Thinking in C++. Volume 1*". 2nd Edition. Prentice Hall, 2000. Online version http://www.etsimo.uniovi.es/eckel/ [Última vez visitado, 10-3-2000].

Eckel, Bruce. "*Thinking in C++. Volume 2: Standard Libraries & Advanced Topics*". 2nd Edition. Prentice Hall, 2000. Online version http://www.etsimo.uniovi.es/eckel/ [Última vez visitado, 10-3-2000].

García Peñalvo, Francisco José. "*Lenguaje de Programación C++*". Notas del curso impartido para la A.L.I (Asociación de Licenciados, Doctores e Ingenieros en Informática). Facultad de Ciencias de la Universidad de Valladolid. Revisión II – Febrero, 1995.

García Peñalvo, Francisco José. "*Apuntes del Lenguaje de Programación C++*". Asignatura de Programación Avanzada, 3er curso de la Ingeniería Técnica en Informática de Gestión de la Universidad de Burgos. (Revisión I – Octubre, 1997).

Hernández Orallo, Enrique y Hernández Orallo, José. "*Programación en C++*". Paraninfo, 1993.

Inprise Corporation. "*Borland C++ Builder 4. Guía del Desarrollador*". Inprise Corporation, 1999.

Joyanes Aguilar, Luis. "*C++ a su Alcance. Un Enfoque Orientado a Objetos*". McGraw-Hill, 1994.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Kruglinski, David J. "*Programación Avanzada con Microsoft Visual C++ 5*". Serie de Programación Microsoft. Microsoft Press. McGraw-Hill, 1998.

Ladd, Scott Robert. "*C++ Techniques and Applications*". M&T Books, 1990.

Leslie, Martin. "*C++ Programming Reference*". Release 0.1. October, 1998.

Lippman, S. and Lajoie, J. "*C++ Primer*". 3rd Edition. Addison-Wesley, 1998.

Microsoft Corporation. "*C++ Tutorial. Microsoft C/C++*". Microsoft Corporation, 1992.

Microsoft Corporation. "*Microsoft Visual C++ 6.0 Documentation*". Microsoft Developer Network 6.0a. 1998.

Nolden, Ralph. "*The User Manual to KDevelop. The Reference Guide to the KDevelop Integrated Development Environment for Unix Systems. Version 1.0*". Version 2.2. The KDevelop Team. http://www.kdevelop.org. [Última vez visitado, 21-3-2000[. July, 1999.

Pappas, Chris H. and Murray, William H. III. "*Manual de Borland C++*". McGraw-Hill, 1993.

Schildt, Herbert. "*Turbo C/C++ 3.1. Manual de Referencia*". Osborne McGraw-Hill, 1994.

Schildt, Herbert. "*Programación en C y C++ en Windows 95*". Osborne McGraw-Hill, 1995.

Stroustrup, Bjarne. "*The Design and Evolution of C++*". Addison-Wesley, 1994. Reprinted with corrections in April, 1995.

Stroustrup, Bjarne. "*El Lenguaje de Programación C++*". 3ª Edición. Addison-Wesley, 1998.

Williams, Mickey. "*Visual C++ 4*". Prentice-Hall Hispanoamericana, 1996.

Yao, Paul and Leinecker, Richard C. "*Todo Visual C++ 5. IDG Bible*". Inforbook's, 1998.

Bibliografía Eiffel

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Meyer, Bertrand. "*Object Oriented Software Construction*". Prentice Hall, 1988.

Meyer, Bertrand. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

Meyer, Bertrand. "*Reusable Software. The Base Object-Oriented Component Libraries*". The Object-Oriented Series. Prentice Hall, 1994.

Meyer, Bertrand. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999.

Bibliografía de Java

Arnold, Ken and Gosling, James. "*El Lenguaje de Programación Java*". Addison-Wesley/Domo, 1997.

Bishop, Judy. "*Java. Fundamentos de Programación*". 2ª Edición. Addison-Wesley, 1999.

Cadenhead, Rogers. "*Teach Yourself JAVA 1.2 in 24 hours*". Sams, 1999.

Davis, Mark. "*Java Cookbook. Porting C++ to Java*". IBM. http://www.ibm.com/java/education/portingc/index.html [Última vez visitado, 13-3-2000]. March, 1997.

Deitel, H. M. and Deitel, P. J. "*Java How To Program*". 3rd Edition. Prentice Hall, 1999.

Eckel, Bruce. "*Thinking in Java*". 2nd Edition. Prentice Hall, 2000. Online version http://www.etsimo.uniovi.es/eckel/ [Última vez visitado, 10-3-2000].

Farley, Jim. "*Java Distributed Computing*". O'Reilly & Associates, Inc., 1998.

Jaworski, Jamie. "*Java 1.2 al Descubierto*". Prentice Hall, 1999.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Lemay, Laura Lemay and Cadenhead, Rogers. "*Teach Yourself Java 2 Platform in 21 Days: Professional Reference Edition*". Sams Teach Yourself in 21 Days Series. Sams, 1999.

Manzanedo del Campo, Miguel Ángel (director) y García Peñalvo, Francisco José (coordinador técnico). "*Guía de Iniciación al Lenguaje Java*". Tutorial subvencionado por la Junta de Castilla y León. Versión 2.0. http://tejo.usal.es/~fgarcia/doc/tuto2/Index.htm [Última vez visitado, 16-3-2000]. Octubre, 1999.

Moreno García, María N., García Peñalvo, Francisco José y García-Bermejo Giner, José Rafael. "*Curso de Iniciación a Java*". Documentación del Curso Extraordinario de la Universidad de Salamanca. Departamento de Informática y Automática. Facultad de Ciencias. Universidad de Salamanca. Octubre, 1999.

Naughton, Patrick. "*Manual de Java*". Osborne McGraw-Hill, 1996.

Rinehart, Martin. "*Desarrollo de Bases de Datos en Java*". Osborne McGraw-Hill, 1998.

SUN Microsystems. "*Java™ Standard Edition Platform Documentation*". http://java.sun.com/docs/index.html. [Última vez visitado, 16-3-2000]. 2000.

SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

Vanhelsuwé, Laurence, Philips, Ivan, Hsu, Goang-Tay, Sankar, Krishna, Ries, Eric, Rohaly, Tim and Zukowksi, John. "*La Biblia Java*". Anaya-Multimedia, 1997.

Bibliografía de STL

Deitel, H. M. and Deitel, P. J. "*C++ How To Program*". 2nd Edition. Prentice Hall, 1998.

Devis Botella, Ricardo. "*C++. STL, Plantillas, Excepciones, Roles y Objetos*". Paraninfo, 1997.

Eckel, Bruce. "*The STL Made Simple*". http://www.mindview.net/stlsimpl.html. [Última vez visitado, 10-3-2000]. 1996.

Eckel, Bruce. "*Thinking in C++. Volume 2: Standard Libraries & Advanced Topics*". 2nd Edition. Prentice Hall, 2000. Online version http://www.etsimo.uniovi.es/eckel/ [Última vez visitado, 10-3-2000].

Glass, Graham and Schuchert, Brett. "*The STL <Primer>*". Prentice Hall, 1996.

Harvey, David. "*Effective STL*". http://web.ftech.net/~honeyg/articles/eff_stl.htm. [Última vez visitado, 10-3-2000]. June, 1997.

Harvey, David. "*A Tiny STL Primer*". http://web.ftech.net/~honeyg/articles/tiny_stl.htm. [Última vez visitado, 10-3-2000]. November, 1997.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

Khan, Mumit. "*Mumit's STL Newbie Guide*". http://www.xraylith.wisc.edu/~khan/software/stl/STL.newbie.html. [Última vez visitado, 10-3-2000]. 1995.

Kirman, Jak. "*A Modest STL Tutorial*". http://www.cs.brown.edu/people/jak/proglan/stltut/tut.html. [Última vez visitado, 10-11-1999]. January 1998.

Plauger, P. J. "*Dinkum C++ Library Reference*". http://www.dinkumware.com/htm_cpl/index.html. [Última vez visitado, 10-3-2000]. 1996.

Rensselaer Polytechnic Institute. "*Standard Template Library Reference*". Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/~musser/stl. [Última vez visitado, 6-3-2000]. 1994.

Silicon Graphics Computer Systems, Inc. "*Standard Template Library Programmer's Guide*". Silicon Graphics Computer Systems, Inc. http://www.sgi.com/Technology/STL/. [Última vez visitado, 10-3-2000]. 1999.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". STL Documentation. October. 1995.

Weidl, Johannes. "*The Standard Template Library Tutorial*". Technical Report 184.437 Wahlfachpraktikum (10.0). Information Systems Institute. Distributed Systems Department. Technical University Vienna. http://www.infosys.tuwien.ac.at/Research/Component/Tuturial/prwmain.htm. [Última vez visitado, 6-3-2000]. April, 1996.

Zalewski, Kenny. "*Standard Template Library Online Reference Home Page*". Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/projects/STL/htdocs/stl.html. [Última vez visitado, 10-3-2000]. May, 1996.

Entornos de desarrollo (C++)

- **Borland C++ Builder**
  - **Versión actual:** 4
  - **Descripción:** Sistemas de desarrollo C/C++ de 32 bits para Windows 9x/NT.
  - **Copyright:** Inprise Inc.
- **DJGPP V2**
  - **Versión actual:** 2.03
  - **Descripción:** Sistema de desarrollo C/C++ de 32-bits para procesadores Intel 80386 (o superiores) bajo sistema operativo DOS (Windows 9x/NT).
  - **URL:** http://www.delorie.com/djgpp/
  - **Copyright:** GNU
- **GNU C/C++ for Linux (gcc – egcs)**
  - **Versión actual:** gcc 2.95.2 – egcs 1.1.2
  - **Descripción**: Compilador de C/C++ para Linux.
  - **URL:** http://www.gnu.org/software/gcc/gcc.html
  - **Copyright:** GNU
- **KDevelop**
  - **Versión actual:** 1.1
  - **Descripción:** Entorno de desarrollo visual de C/C++ para Unix/X11/KDE.
  - **URL:** http://www.kdevelop.org/
  - **Copyright:** GNU
- **Microsoft Visual C++**
  - **Versión actual:** 6.0
  - **Descripción:** Sistema de desarrollo C/C++ de 32 bits integrado en Visual Studio 6.0. Bajo sistema operativo Windows 9x/NT.
  - **Copyright:** Microsoft.

Entornos de desarrollo (Java)

- **JAVA™ 2 SDK, Standard Edition**
  - **Versión actual:** 1.2.2
  - **Descripción:** Sistema de desarrollo Java para Windows 9x/NT, Solaris y Linux.
  - **URL:** http://java.sun.com/products/jdk/1.2/
  - **Copyright:** Sun Microsystems.

Herramientas CASE

- **ArgoUML**
  - **Versión actual:** 0.7.0
  - **Descripción:** Herramienta para el modelado de sistemas software con UML. Multiplataforma.
  - **URL:** http://argouml.tigris.org/
  - **Copyright:** GNU
- **Rational Rose**
  - **Versión actual:** Rose 2000
  - **Descripción:** Herramienta para el modelado de sistemas software con UML. Disponible para Windows 9x/NT y algunas versiones de UNIX
  - **URL:** http://www.rational.com
  - **Copyright:** Rational Software Corporation. Versiones de demostración totalmente funcionales, pero limitadas en el tiempo de uso.

5.4 Fuentes de información sobre Ingeniería del Software y tecnología de Objetos

Este apartado ofrece una panorámica de las principales fuentes de información sobre la materia objeto de la plaza a concurso. Se incluye información de publicaciones periódicas, jornadas y congresos, sitios web y grupos de noticias.

5.4.1 Revistas

Las revistas que se enumeran a continuación tienen en sus contenidos artículos relacionados con la Ingeniería del Software y la Orientación a Objetos.

- **ACM Computing Surveys**. Disponible desde el volumen 17 (1985) en la biblioteca digital de ACM (http://www.acm.org/pubs/contents/journals/surveys/).
- **ACM SIGSOFT (Software Engineering Notes)**. (http://www.acm.org/sigs/sigsoft/SEN/).
- **ACM TOSEM (Transaction On Software Engineering and Methodology)**. Disponible desde el volumen 1 (1992) en la biblioteca digital de ACM (http://www.acm.org/pubs/contents/journals/tosem/).
- **Communications of the ACM**. Disponible desde el volumen 28 (1985) en la biblioteca digital de ACM (http://www.acm.org/pubs/contents/journals/cacm/).
- **Crosstalk. The Journal of Defense Software Engineering**. Disponible desde el volumen 7 (1994) en http://stsc.hill.af.mil/crosstalk/xtalk.asp.
- **C/C++ Users Journal** (http://www.cuj.com).
- **C++ Report**. Índices y algunos artículos en línea en http://www.creport.com/.
- **Dr. Dobb's Journal** (http://www.ddj.com).
- **IBM System Journal** (http://www.research.ibm.com/journal/).
- **IEEE Computer**. Disponible desde el volumen 28 (1995) en la biblioteca digital de IEEE (http://computer.org/computer/).
- **IEEE Internet Computing**. Disponible desde el volumen 1 (1997) en la biblioteca digital de IEEE (http://computer.org/internet/).
- **IEEE ITPro**. Disponible desde el volumen 1 (1999) en la biblioteca digital de IEEE (http://computer.org/itpro/).
- **IEEE Software**. Disponible desde el volumen 12 (1995) en la biblioteca digital de IEEE (http://computer.org/software/).
- **IEEE TSE (Transactions on Software Engineering)**. Disponible desde el volumen 21 (1995) en la biblioteca digital de IEEE (http://computer.org/tse/).
- **International Journal of Software Engineering and Knowledge Representation** (World Scientific Publishing Company).
- **Java Report**. Índices y algunos artículos en línea en http://www.javareport.com/.
- **JavaWorld**. Revista online en http://www.javaworld.com/.
- **JOOP (Journal of Object-Oriented Programming)**. Índices y algunos artículos en línea en http://www.joopmag.com/.
- **Novática** (http://www.ati.es/).

- **Rose Architect Magazine** (http://www.researchitect.com).
- **SEI Interactive** (http://interactive.sei.cmu.edu/home.htm).
- **Software Development**. Artículos en línea en (http://www.sdmagazine.com/).
- **Software--Practice and Experience** (John Wiley & Sons).
- **Theory and Practice of Object Systems – TAPOS** (John Wiley & Sons).

5.4.2 Jornadas y congresos

5.4.2.1 Nacionales

- **Congreso de Tecnología de Objetos.**
- **Jornadas de Ingeniería del Software – JIS**. Vienen realizándose anualmente desde 1996 [Toro, 1996], [Díaz y Lopistéguy, 1997], [Toval y Nicolás, 1998], [Botella et al., 1999]. Se ha consolidado como el foro nacional de encuentro y discusión de los Grupos de Investigación y Docencia en Ingeniería del Software. En la última edición celebrada en noviembre de 1999 en Cáceres, se unieron las Jornadas de Ingeniería del Software y de Bases de Datos.
- **Jornadas de Investigación y Docencia en Bases de Datos – JIDBD**. Vienen celebrándose anualmente desde 1996.
- **Jornadas de Trabajo MENHIR**. El proyecto coordinado MENHIR (proyecto CICYT TIC97-0593-C05-05) ha dado como resultado una serie de reuniones entre los Grupos de investigación integrantes del mismo desde noviembre de 1997 hasta marzo de 2000.
- **Jornadas sobre Calidad del Software**.

5.4.2.2 Internacionales

- **ACM Annual Symposium on Principles of Programming Languages – POPL.**
- **ACM Conference on Object Oriented Programming Systems Languages and Aplications – OOPSLA.**
- **ACM Foundations of Software Engineering – ACM SIGSOFT.**
- **Asia Pacific Conference on Pattern Languages of Programs – KoalaPLoP.**
- **Conference on Pattern Languages of Programs – PLoP.**
- **European Conference on Object-Oriented Programming – ECOOP.**
- **European Conference on Pattern Languages of Programming and Computing – EuroPLoP.**
- **IEEE Asia Pacific Software Engineering Conference – APSEC.**
- **IEEE Australian Software Engineering Conference – ASWEC.**
- **IEEE Basque International Workshop on Information Technology – BIWIT.**
- **IEEE Conference on Software Engineering Education & Training - CSEE&T.**
- **IEEE European Conference on Software Maintenance and Reengineering – CSMR.**
- **IEEE International Conference on Requirements Engineering – ICRE.**
- **IEEE International Conference on Software Maintenance – ICSM.**
- **IEEE International Conference on Software Reuse – ICSR.**
- **IEEE International Symposium on Software Engineering Standards – ISESS.**

- **IEEE International Symposium on Object-Oriented Real-Time Distributed Computing – ISORC.**
- **IEEE International Symposium on Software Engineering forParallel and Distributed Systems – PDSE.**
- **IEEE International Symposium on Requirements Engineering – RE.**
- **IEEE Real-Time Technology and Applications Symposium – RTAS.**
- **IEEE International Conference on Real-Time Computing Systems and Applications – RTCSA.**
- **IEEE Real-Time Systems Symposium – RTSS.**
- **International Conference on Software Engineering – ICSE.**
- **Jornadas Iberoamericanas de Ingeniería del Requisitos y Ambientes Software – IDEAS**.
- **Technology of Object-Oriented Languages and Systems – TOOLS (TOOLS Europe; TOOLS USA; TOOLS Pacific).**
- **<<UML>> Conference.**

5.4.3 Sitios web

Son innumerables los servidores en Internet con información relacionada con la Ingeniería del Software y la Orientación a Objetos. Se pueden encontrar libros, documentos, herramientas, lenguajes, datos reales de proyectos a gran escala, estándares...

Ofrecer un listado exhaustivo de todos estos lugares sería impracticable, y su utilidad sería relativa por la volatilidad que presentan algunos de ellos. Por este motivo se ha hecho una selección de aquellos sitios web más importantes y más estables.

- **Association for Computing Machinery (ACM):** http://www.acm.org. Fundada en 1947 fue la primera sociedad científica y de educación del mundo. El portal de información que presenta es impresionante, tanto en cuanto a enlaces de interés, grupos de trabajo, documentos electrónicos, conferencias como por su biblioteca digital conteniendo revistas y actas de congresos.
- **CASE:** http://osiris.sunderland.ac.uk/sst/casehome.html. Enlaces a varias herramientas CASE de libre difusión.
- **Cetus Links - Object-Orientation:** http://www.rhein-neckar.de/~cetus/software.html. La colección de enlaces más completa sobre Orientación a Objetos. A fecha del 27 de febrero de 2000 se tenían contabilizados 18558 enlaces.
- **European Software Institute (ESI):** http://www.esi.es. El Instituto del Software Europeo tiene su sede en Bilbao (España). Dispone tanto de documentación privada para los miembros del instituto como documentación pública con los análisis de proyectos, necesidades de empresas y software europeo.
- **Guide to the Software Engineering Body of Knowledge (SWEBOK):** http://www.swebok.org/. Proyecto para establecer un cuerpo de conocimiento común para la Ingeniería del Software.
- **Institute of Electrical and Electronics Engineers (IEEE):** http://www.ieee.org. Otra prestigiosa organización compuesta por diversas sociedades, donde la que

más relación tiene con los temas abordados en el presente Proyecto Docente es la IEEE Computer Society (http://computer.org). A semejanza de ACM, ofrece información sobre conferencias, estándares, educación y mantiene otra biblioteca digital con revistas y actas de congresos.

- **Object Management Group (OMG):** http://www.omg.org. Es un consorcio internacional de industrias con el fin de promover el uso de la Orientación a Objetos en la Ingeniería del Software. A diferencia de organizaciones como ISO o IEEE, OMG desarrolla estándares de "*facto*" como consenso entre las empresas que la forman. Dicho servidor ofrece publicaciones electrónicas y enlaces a estándares y herramientas del sector relacionado con la tecnología de objetos.
- **Object-Oriented Information Sources:** http://iamwww.unibe.ch/~scg/OOinfo/index.html. Colección de enlaces catalogados por temas sobre la tecnología de objetos.
- **Rational Software Corporation:** http://www.rational.com. Sitio web de la empresa responsable de UML. En esta dirección se tiene valiosa información sobre UML y RUP (documentos, informes, artículos, presentaciones, bibliografía recomendada...). Además, se pueden obtener versiones de demostración de diferentes herramientas que comercializan, siendo Rational Rose 2000 la más difundida.
- **R.S. Pressman & Associates, Inc.:** http://www.rspa.com. Bajo la dirección de Roger S. Pressman y la difusión internacional de su libro [Pressman, 1997], cuya quinta edición está actualmente en desarrollo, aparece una empresa de consultoría en Ingeniería del Software. Lo más interesante que ofrece esta dirección es un portal que da entrada a otras fuentes de información relacionadas con cada uno de los capítulos tratados en su libro.
- **Software Engineering Institute (SEI):** http://www.sei.cmu.edu. El Instituto de Ingeniería del Software en la Universidad Carnegie Mellon, es uno de los lugares más activos en pro de la Ingeniería del Software. Se pueden encontrar documentos asociados a módulos curriculares en Ingeniería del Software, informes técnicos sobre diferentes áreas de la Ingeniería del Software, una revista en línea... Incluye además enlaces a otras organizaciones relacionadas con la Ingeniería del Software.
- **Software Engineering Laboratory (SEL):** http://fdd.gsfc.nasa.gov. El Laboratorio de Ingeniería del Software, en el que participan la Universidad de Maryland y el Centro de Vuelo Espacial de la NASA, analiza los proyectos desarrollados en dicho centro y propone metodologías para la mejora del proceso de desarrollo. Se ofrecen además datos relativos a proyectos reales.
- **The Pedagogical Patterns Project:** http://www-lifia.info.unlp.edu.ar/ppp/. Proyecto de patrones pedagógicos. Aplicación de la teoría de patrones al terreno de la pedagogía.

5.4.4 Grupos de noticias

5.4.4.1 Ingeniería del Software

- comp.software-eng
- comp.specification
- comp.testing

5.4.4.2 Orientación a Objetos

- comp-databases.object
- comp.object
- comp.object.corba

5.4.4.3 C++

- comp.lang.c++
- comp.lang.c++.moderated
- comp.std.c++
- es.comp.lenguajes.c++
- gnu.g++.help

5.5 Referencias

[Abran et al., 1999] Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.5. ACM/IEEE-CS, October, 1999.

[Aldis, 1995] Aldis, Margaret. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

[Alexander, 1964] Alexander, Christopher. "*Notes on the Synthesis of Form*". Harvard University Press, 1964.

[Alexander, 1979] Alexander, Christopher. "*The Timeless Way of Building*". The Oxford University Press, 1979.

[Alexander et al., 1975] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M. and Angel, S. "*The Oregon Experiment*". Oxford University Press, 1975.

[Alexander et al., 1977] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. "*A Pattern Language*". Oxford University Press, 1977.

[Ancona et al., 1992] Ancona, D., Astesiano, E. and Zucca, E. "*Towards a Classification of Inheritance Relations*". Technical Report, Dipartimento di Informatica e Scienze dell'Informazione, Genova. 1992.

[Apple, 1989] Apple Computer Inc. "*Macintosh Programmer's Workshop Pascal 3.0 Reference*". Apple Computer, 1989.

[Appleton, 2000] Appleton, Brad. "*Patterns and Software: Essential Concepts and Terminology*". http://www.enteract.com/~bradapp/docs/patterns-intro.html [Última vez visitado, 15-3-2000]. February, 2000.

[Arnold and Gosling, 1996] Arnold, Ken and Gosling, James. "*The Java Programming Language*". Addison-Wesley, 1996.

[Arnold and Gosling, 1997] Arnold, Ken and Gosling, James. "*The Java Programming Language*". 2nd Edition. Addison-Wesley, 1997.

[Astrachan et al., 1998] Astrachan, Owen, Berry, Geoffrey, Cox, Landon and Mitchener, Garret. "*Design Patterns: An Essential Component of CS Curricula*". In Proceedings of

the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 153-160. 1998.

[Beck and Cunningham, 1988] Beck, Kent and Cunningham, Ward. "*Using a Pattern Language for Programming*". In Addendum to the Proceedings of OOPSLA'87. ACM SIGPLAN Notices, 23(5). May, 1988.

[Beck and Cunningham, 1989] Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

[Bellin, 1992] Bellin, D. "*A Seminar Course in Object-Oriented Programming*". SIGCSE Bulletin, 24(1):134-137. 1992.

[Biggerstaff, 1992] Biggerstaff, T. J. "*An Assessment and Analysis of Software Reuse*". Advances in Computers. Vol. 34:1-57. 1992.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Booch and Rumbaugh, 1995] Booch, Grady and Rumbaugh, James. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

[Booch et al., 1996] Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

[Booch et al., 1999] Booch, Grady, Rumbaugh, James and Jacobson, Ivar. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

[Botella et al., 1999] Botella, Pere, Hernández, Juan y Saltor, Fèlix (editores). "*Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*". Grupo de paralelismo del Departamento de Informática de la Universidad de Extremadura. Cáceres, 24-26 de noviembre de 1999.

[Brackett, 1990] Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1990.

[Brodsky, 1999] Brodsky, S. "*XMI Opens Application Interchange*". White Paper. IBM. March, 1999.

[Brown et al., 1998] Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". Wiley & Sons, 1998.

[Bruegge and Dutoit, 2000] Bruegge, Bernd and Dutoit, Allen H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

[Budd, 1991] Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Budgen, 1999] Budgen, David. "*Software Design Methods: Life Belt or Leg Iron*". IEEE Software, 16(5):133-136. September/October, 1999.

[Buschmann et al., 1996] Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad, Peter and Stal, Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

[Cardelli and Wegner, 1985] Cardelli, L. and Wegner, P. "*On Understanding Types, Data Abstraction and Polymorphism*". Computing Surveys, 17(4):471-523. December, 1985.

[Cardelli et al., 1989] Cardelli, L., Donahue, J., Glassman, L., Jordan, M., Kalsow, B. and Nelson, G. "*Modula-3 Report (revised)*". Technical Report 52, Systems Research Center, Digital Equipment Corporation, Palo Alto. 1989.

[Carsí, 1999] Carsí, J. A. "*OASIS como Marco Conceptual para la Evolución de Software*". Tesis Doctoral. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. 1999.

[Clancy and Linn, 1999] Clancy, Michael J. and Linn, Marcia C. "*Patterns and Pedagogy*". In Proceedings of the thirtieth SIGCSE technical symposium on Computer science education, SIGCSE '99. (March 24-28, 1999, New Orleans, LA - USA). Pages 37-42. ACM. 1999.

[Coplien, 1992] Coplien, James O. "*Advanced C++ Programming Styles and Idioms*", Addison-Wesley, 1992.

[Coplien, 1998] Coplien, James O. "*A Pattern Definition - Software Patterns*". http://hillside.net/patterns/definition.html. 1998.

[Crespo, 2000] Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

[Cybulski, and Linden, 2000] Cybulski, Jacob L. and Linden, Tanya. "*Teaching Systems Analysis and Design Using Multimedia and Patterns*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 113-122. IEEE Press, 2000.

[Chen, 1976] Chen, Peter. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March, 1976.

[Danforth and Tomlinson, 1988] Danforth, S. and Tomlinson, C. "*Type Theories and Object-Oriented Programming*". ACM Computing Surveys, 20(1):29-72, 1988.

[Davis, 1983] Davis, William S. "*Tools and Techniques*". Addison-Wesley, 1983.

[DeClue, 1996] DeClue, Tim. "*Object-Oriented and the Principles of Learning Theory: A New Look at Problems and Benefits*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 232-236. ACM. 1996.

[DeMarco, 1979] DeMarco, Tom "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

[Denning, 1992] Denning, Peter J. "*Educating a New Engineer*". Communications of the ACM, 35(12). December, 1992.

[Deveaux et al., 1999] Deveaux, Daniel, Fleurquin, Regis and Frison, Patrice. "*Software Engineering Teaching: A 'Docware' Approach*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 163-166. ACM. 1999.

[Díaz y Lopistéguy, 1997] Díaz, Oscar y Lopistéguy, Philippe (editores). "*Actas de las II Jornadas de Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad del País Vasco/Euskal Herriko Unibertsitatea. Donosita-San Sebastián, 2-5 de septiembre de 1997.

[Donadi, 1992] Donadi, Mahesh H. "*Teaching Practical Object-Oriented Software Engineering*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 251-256. ACM. 1992.

[Durán y Bernárdez, 1999a] Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/norma/analisis-2.zip. [Última vez visitado, 10-3-2000]. Sevilla, 29 de noviembre de 1999.

[Durán y Bernárdez, 1999b] Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/norma/elicitacion.zip. [Última vez visitado, 10-3-2000]. Sevilla, 18 de octubre de 1999.

[EIA CDIF Division, 1996] EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

[Fairley, 1985] Fairley, Richard. "*Software Engineering Concepts*". McGraw-Hill, 1985.

[Fell et al., 1996] Fell, Harriet J., Proulx, Viera K. and Casey, John. "*Writing across the Computer Science Curriculum*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 204-209. ACM. 1996.

[Fell et al., 1998] Fell, Harriet J., Proulx, Viera K. and Rasala, Richard. "*Scaling: A Design Pattern in Introductory Computer Science Courses*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 326-330. 1998.

[Firesmith et al., 1998] Firesmith, Donald, Henderson-Sellers, Brian and Graham, Ian. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

[Fisher, 1991] Fisher, Alan S. "*CASE: Using Software Development Tools*". 2nd Edition. John Wiley & Sons, 1991.

[Fowler, 1996] Fowler, Martin. "*Analysis Patterns: Reusable Object Models*". Object Technology Series. Addison-Wesley, 1996.

[Fowler and Scott, 1997] Fowler, Martin and Scott, Kendall. "*UML Distilled. Applying the Standard Object Modeling Language*". Object Technology Series. Addison Wesley, 1997.

[Fowler and Scott, 2000] Fowler, Martin and Scott, Kendall. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd Edition. Object Technology Series. Addison Wesley, 2000.

[Gamma, 1991] Gamma, Erich. "*Objektorientierte Software-Entwicklung am Beispiel von ET++: Klassenbibliothek, Werkzeuge, Desing*". Dissertation, Universität Zürich, 1991.

[Gamma et al., 1993] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns: Abstraction and Reuse of Object-Oriented Design*". In Proceedings of the 7th European Conference in Object Oriented Programming – ECOOP'93. Nierstrasz, O. M. editor. Pages 406-431. Springer Verlag, 1993.

[Gamma et al., 1995] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

[Gane and Sarson, 1981] Gane, Chris and Sarson, Trish. "*Análisis Estructurado de Sistemas*". Ateneo, 1981.

[Garbajosa y Bonilla, 1995] Garbajosa, Juan y Bonilla, A. "*Integración de Herramientas CASE*". Capítulo 22 en [Piattini y Daryanani, 1995]. 1995.

[García, 1998] García Peñalvo, Francisco José. "*Patrones. De Alexander a la Tecnología de Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(10):44-52. Noviembre, 1998. (También disponible en http://tejo.usal.es/~fgarcia/doc/patrones1.pdf).

[García, 1999] García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

[García, 2000] García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[García et al., 1997] García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report TR-GIRO-01-97V2.1.1. Versión 2.1.1. Universidad de Valladolid. 1997.

[García et al., 2000] García Peñalvo, Francisco José, Moreno García, María N., García-Bermejo Giner, José Rafael y Luis Reboredo, Ana de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García y Pardo, 1998] García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Gelfand et al., 1998] Gelfand, Natasha, Goodrich, Michael T. and Tamassia, Roberto. "*Teaching Data Structure Design Patterns*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 331-335. 1998.

[Gersting, 1994] Gersting, Judith L. "*A Software Engineering 'Frosting' on a Traditional CS-1 Course*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 233-237. ACM. 1994.

[Ghezzi and Jazayeri, 1998] Ghezzi, C. and Jazayeri, M. "*Programming Languages Concepts*". 3rd Edition. John Wiley & Sons, 1998.

[Glass, 1996] Glass, Robert L. "*The Relationship between Theory and Practice in Software Engineering*". Communications of the ACM, 39(11):11-13. November, 1996.

[Goguen, 1984] Goguen, J. "*Parametrized Programming*". IEEE Transactions on Software Engineering, 10(5):528-543, 1984.

[Goldberg and Robson, 1983] Goldberg, Adele and Robson, David. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd edition. Addison-Wesley, 1994.

[Graham, 1995] Graham, Ian M. "*Migrating to Object Technology*". Addison-Wesley, 1995.

[Graham et al., 1997] Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

[Granger and Little, 1996] Granger, Mary J. and Little, Joyce Currie. "*Integrating CASE Tools into the CS/CIS Curriculum*". In Proceedings of the conference on Integrating technology into computer science education, ITiCSE '96. (June 2-6, 1996, Barcelona, Spain). Pages 130-132. ACM, 1996.

[Griss, 1995] Griss, Martin L. "*The Architecture and Processes for Systematic OO Reuse Factory*". In Proceedings of the 7th Workshop on Institutionalizing Software Reuse (WISR-7), Andersen Consulting's Center for Professional Education, St. Charles, Illinois, 28-30 August 1995.

[Guerraoui et al., 1996] Guerraoui, R. (editor), Aksit, M., Black, A., Cardelli, L., Cointe, P., Coplien, J., Kiczales, G., Lea, D., Madsen, O., Magnusson, B., Meseguer, J., Moessenboeck, H., Palsberg, J. and Schmidt, D. "*Strategic Research Directions in Object Oriented Programming*". Technical Report based on position statements held during ACM Workshop on Strategic Directions in Computing. MIT, June, 1996.

[Halbert and O'Brien, 1987] Halbert, D. and O'Brien, P. "*Using Types and Inheritance in Object-Oriented Programs*". IEEE Software, pages 71-79. 1987.

[Hatley and Pirbhai, 1987] Hatley, Derek J. and Pirbhai, Imtiaz. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

[Henderson-Sellers, 1997] Henderson-Sellers, B. "*OPEN Relationships – Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December, 1997.

[Henderson-Sellers and Edwards, 1990] Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

[Henderson-Sellers and Edwards, 1994] Henderson-Sellers, B. and Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 4: Resource and Technique Standards*". IEEE Computer Society Press, 1999.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jackson, 1995] Jackson, Michael. "*Critical Reading for Software Developers*". IEEE Software, 12(6):103-104. November, 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Jacobson et al., 1999] Jacobson, I., Booch, G. and Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

[Jézéquel and Meyer, 1997] Jézéquel, Jean-Marc and Meyer, Bertrand. "*Design by Contract: The Lessons of Ariane*". IEEE Computer, 30(1):129-130. January, 1997.

[Joyanes, 1998] Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Joyner, 1999] Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

[Karlsson, 1995] Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

[Kernighan and Ritchie, 1988] Kernighan, Brian W. and Ritchie, Dennis M. "*The C Programming Language*". Prentice-Hall, 1988.

[Knudsen and Madsen, 1988] Knudsen, J. L. and Madsen, O. L. "*Teaching Object-Oriented Programming Languages*". In Proceedings of ECOOP'98. Springer-Verlag. Pages 21-40. August, 1988.

[Kölling, 1999a] Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 1: Languages*". Journal of Object-Oriented Programming, 11(8):8-15. January, 1999.

[Kölling, 1999b] Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 2: Environments*". Journal of Object-Oriented Programming, 11(9):6-12. February, 1999.

[Krasner and Pope, 1988] Krasner, G. E. and Pope, S. T. "*A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*". Journal of Object-Oriented Programming, 1(3):26-49. August-September, 1988.

[Kruglinski, 1995] Kruglinski, D. "*Inside Visual C++*". Microsoft Press, 1995.

[Lalonde and Pugh, 1991] Lalonde, Wilf R. and Pugh, John R. "*Inside Smalltalk*". Vol. 2. Prentice-Hall, 1991.

[Lea, 1994] Lea, Doug. "*Christopher Alexander: An Introduction for Object-Oriented Designers*". Software Engineering Notes, ACM SIGSOFT, 19(1):39-46, January 1994 (Online version: http://g.oswego.edu/dl/ca/ca/ca.html).

[Letelier et al., 1998] Letelier, P., Ramos, I., Sánchez, P., y Pastor, O. "*OASIS Versión 3.0. Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-98.4011. 1998.

[Lilly, 1996] Lilly, Susan. "*Patterns for Pedagogy*". Object Magazine, 5(8):93-96. January, 1996.

[Lippman, 1989] Lippman, S. "*C++ Primer*". Addison-Wesley, 1989.

[Liskov, 1987] Liskov, Barbara. "*Data Abstraction and Hierarchy*". In Addedum to Proceedings of OOPSLA'87. Pages 17-35. ACM Press, 1987.

[Liskov, 1993] Liskov, Barbara. "*A History of CLU*". In Proceedings of the second ACM SIGPLAN conference on History of programming languages – HOPL II. (April 20 - 23, 1993, Cambridge United States). ACM SIGPLAN Notices, 28(3):133-147. March, 1993.

[Luker, 1994] Luker, P. "*There's More to OOP than Syntax!*". SIGCSE Bulletin, 26(1):56-60. 1994.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[Marqués, 1995] Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

[Marqués, 1999] Marqués Corral, José Manuel. "*Estándares de documentación. Laboratorio de Ingeniería del Software I. Ingeniería Técnica en Informática de Sistemas de la Universidad de Valladolid*". Departamento de Informática de la Universidad de Valladolid. Septiembre, 1999.

[Martin, 1996a] Martin, Robert C. "*The Open Closed Principle*". C++ Report, 8(1). January, 1996.

[Martin, 1996b] Martin, Robert C. "*The Liskov Substitution Principle*". C++ Report, 8(3). March, 1996.

[Martin, 1996c] Martin, Robert C. "*The Dependency Inversion Principle*". C++ Report, 8(5). May, 1996.

[Martin, 1996d] Martin, Robert C. "*The Interface Segregation Principle*". C++ Report, 8(7). July-August, 1996.

[Martin, 1996e] Martin, Robert C. "*Granularity*". C++ Report, 8(10). November-December, 1996.

[Martin, 1997a] Martin, Robert C. "*Principles of OOD*". OMA. 1997.

[Martin, 1997b] Martin, Robert C. "*Stability*". C++ Report, 9(2). February, 1997.

[Martin and Odell, 1995] Martin, James and Odell, James J. "*Object-Oriented Methods: A Foundation*". Prentice Hall, 1995.

[Martin and Odell, 1996] Martin, James and Odell, James J. "*Object-Oriented Methods: Pragmatic Considerations*". Prentice Hall, 1996.

[Martin and Odell, 1998] Martin, James and Odell, James J. "*Object-Oriented Methods: A Foundation, UML Edition*". 2nd Edition. Prentice Hall, 1998.

[McCauley et al., 1996] McCauley, Renée A., Jackson, Ursula and Manaris, Bill. "*Documentation Standards in the Undergraduate Computer Science Curriculum*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 242-246. ACM. 1996.

[McClure, 1996] McClure, Carma. "*Experiences from the OO Playing Field*". The Object World Insider, 2(4):1-3. 1996.

[McClure, 1997] McClure, Carma. "*Software Reuse Techniques: Adding Reuse to the System Development Process*". Prentice Hall, 1997.

[McDonald and McDonald, 1993] McDonald, Gary and McDonald, Merry. "*Developing Oral Communication Skill of Computer Science Undergraduates*". In Proceedings of the twenty-fourth SIGCSE technical symposium on Computer Science Education, SIGCSE '93. (Feb. 18-19, 1993, Indianapolis, IN, USA). Pages 279-282. ACM. 1993.

[Meyer, 1988] Meyer, Bertrand. "*Object Oriented Software Construction*". Prentice Hall, 1988.

[Meyer, 1992] Meyer, Bertrand. "*Eiffel: The Language*". Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

[Meyer, 1994] Meyer, Bertrand. "*Reusable Software. The Base Object-Oriented Component Libraries*". The Object-Oriented Series. Prentice Hall, 1994.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd edition. Prentice Hall, 1997.

[Miguel y Piattini, 1993] Miguel, Adoración de y Piattini, Mario G. "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al ModeloRelacional*". Ra-ma, 1993.

[Minsky, 1981] Minsky, M. "*A Framework for Representing Knowledge*". In Haugeland, J. *Mind Design*. MIT Press, 1981.

[Moitra, 1999] Moitra, Deependra. "*Software Engineering in the Small. Practical Software Engineering and Management*". IEEE Computer, 32(10):39-40. October, 1999.

[Monarchi and Puhr, 1992] Monarchi, David E. and Puhr, Gretchen I. "*A Research Typology for Object-Oriented Analysis and Design*". Communicatios of the ACM, 35(9):35-47. September, 1992.

[Musser and Saini, 1996] Musser, D. and Saini, A. "*STL Tutorial and Reference Guide*". Addison-Wesley, 1996.

[Musser and Stepanov, 1987] Musser, D. R. and Stepanov, A. A. "*A Library of Generic Algorithms in Ada*" Proceedings of 1987 ACM SIGAda International Conference, Boston, December, 1987.

[Musser and Stepanov, 1989a] Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". In Proceedings of the First International Joint Conference of ISSAC-88 and AAECC-6. P. Gianni editor. (Rome, Italy, July 4-8, 1988). Published in *Lecture Notes in Computer Science* 358, Pages 13-25. Springer-Verlag, 1989.

[Musser and Stepanov, 1989b] Musser, David R. and Stepanov, Alexander A. "*The Ada Generic Library: Linear List Processing Packages*". Compass Series. Springer-Verlag, 1989.

[Musser and Stepanov, 1994] Musser, David R. and Stepanov, Alexander A. "*Algorithm-Oriented Generic Library*". Software Practice & Experience, 24(7):623-642. July, 1994.

[Musser and Stepanov, 1998] Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". Dagstuhl Seminar on Generic Programming. 1998.

[Nachbar, 1998] Nachbar, Daniel. "*Bringing Real-World Software Development into the Classroom: A Proposed Role for Public Software in Computer Science Education*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 171-175. 1998.

[Nerson, 1992] Nerson, J. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September, 1992.

[Nguyen, 1998] Nguyen, Dung. "*Design Patterns for Data Structures*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 336-340. 1998.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Parnas, 1972] Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

[Parrish et al., 1997] Parrish, Allen, Lester, Cynthia, Cordes, David and Moore, Deanne. "*Assesing Computer Usage Patterns in a Software Development Course*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education, SIGCSE '97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 58-62. ACM. 1997.

[Piattini, 1996] Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

[Piattini y Daryanani, 1995] Piattini Velthuis, Mario G. y Daryanani Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

[Pressman, 1992] Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Prieto, 1999] Prieto Arambillet, Félix. "*Apuntes de la Asignatura Programación III*". Versión 1.0. Departamento de Informática. Universidad de Valladolid. Diciembre, 1999.

[Proto-Patterns, 1999] "*The Pedagogical Patterns Project. Successes in Teaching Object-Technology (PROTO-PATTERNS)*". http://www-lifia.info.unlp.edu.ar/ppp/index.html. [Última vez visitado, 20/8/1999]. July, 1999.

[Quillian, 1967] Quillian M. R. "*Word Concepts: A Theory and Simulation of some Basic Semantic Capabilities*". Behavioural Science, 12:410-30. 1967.

[Rasala, 1997] Rasala, Richard. "*Design Issues in CS Education*". SIGCSE Bulletin. December, 1997.

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Reenskaug et al., 1996] Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

[Rising, 1996] Rising, Linda. "*Design Patterns: Elements of Reusable Architectures*". Annual Review of Communications, Vol. 49:907-909. 1996.

[Rumbaugh, 1987] Rumbaugh, James E. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). ACM. Reprinted in ACM SIGPLAN 22(12):466-481. October, 1987.

[Rumbaugh, 1994a] Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

[Rumbaugh, 1994b] Rumbaugh, James. "*Building Boxes: Composite Objects*". Journal of Object-Oriented Programming (JOOP), 7(7):12-22. November-December, 1994.

[Rumbaugh, 1995a] Rumbaugh, James. "*OMT: The Object Model*". Journal of Object-Oriented Programming (JOOP), 7(8):21-27. January, 1995.

[Rumbaugh, 1995b] Rumbaugh, James. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming (JOOP), 7(9):6-12. February, 1995.

[Rumbaugh, 1995c] Rumbaugh, James. "*OMT: The Functional Model*". Journal of Object-Oriented Programming (JOOP), 8(1):10-14. March-April, 1995.

[Rumbaugh, 1995d] Rumbaugh, James. "*OMT: The Development Process*". Journal of Object-Oriented Programming (JOOP), 8(2):8-16,76. May, 1995.

[Rumbaugh, 1996] Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

[Rumbaugh, 1998] Rumbaugh, James. "*Depending on Collaborations: Dependencies as Contextual Associations*". Journal of Object-Oriented Programming (JOOP), 11(4):5-9. July/August, 1998.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Sakkinen, 1989] Sakkinen, M. "*Disciplined Inheritance*". In Proceedings of ECOOP'89. Cook, S. editor. Pages 39-56. Cambridge University Press, 1989.

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

[Snyder, 1991] Snyder, A. "*Inheritance in Object-Oriented Programming Languages*". In Lenzerini, M., Nardi, D. and Simi, M. editors. *Inheritances Hierarchies in Knowledge Representation and Programming Languages*. Pages 153-172. John Wiley & Sons, 1991.

[Stepanov and Lee, 1994] Stepanov, A. A. and Lee, M. "*The Standard Template Library*". Technical Report HPL-94-34, Hewlett-Packard Laboratories, April 1994.

[Stepanov and Lee, 1995] Stepanov, A. A. and Lee, M. "*The Standard Template Library*". STL Documentation. October. 1995.

[Stroustrup, 1997] Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

[SUN, 2000] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

[Taivalsaari, 1996] Taivalsaari, A. "*On the Notion of Inheritance*". ACM Computing Surveys, 28(3). 1996.

[Tesler, 1993] Tesler, L. "*Object-Oriented Dynamic Languages*". In Proceedings of the Object Expo Conference. July, 1993.

[Tewari, 1995] Tewari, Rajiv. "*Software Reuse and Object-Oriented Software Engineering in the Undergraduate Curriculum*". In Proceedings of the 26th SISCSE technical symposium on Computer Science Education, SIGCSE '95. (March 2-4, 1995, Nashville, TN, USA). Pages 253-257. ACM, 1995.

[Thompson, 1999] Thompson, S. "*Haskell, The Craft of Functional Programming*". Addison-Wesley, 1999.

[Toro, 1996] Toro Bonilla, Miguel (editor). "*Actas de las I Jornadas de Trabajo en Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 14-15 de noviembre de 1996.

[Toval y Nicolás, 1998] Toval Álvarez, Ambrosio y Nicolás Ros, Joaquín (editores). "*Actas de las III Jornadas de Ingeniería del Software*". Departamento de Informática, Lenguajes y Sistemas de la Universidad de Murcia. Murcia, 11-13 de noviembre de 1998.

[Tuya, 1999] Tuya González, Pablo Javier. "*Manual de Procedimientos para las Prácticas de Ingeniería del Software I y II*". Versión 1.2. Universidad de Oviedo. http://opalo.etsiig.uniovi.es/~tuya/is/procedimientos/procs.htm. [Última vez visitado, 1-12-1999]. Octubre, 1999.

[Vaquero, 1999] Vaquero Sánchez, Antonio. "*La Lengua Española en el Contexto Informático*". Revista de Enseñanza y Tecnología. ADIE. (13):5-12. Enero-Abril, 1999.

[Wallingford, 1996] Wallingford, Eugene. "*Toward a First Course Based on Object-Oriented Patterns*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 27-31. ACM. 1996.

[Ward and Mellor, 1985] Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

[Wegner, 1987] Wegner, Peter. "*The Object-Oriented Classification Paradigm in Research Directions on Object-Oriented Programming*". MIT Press, Cambridge, MA, 1987.

[Welch and Strong, 1998] Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

[Wirfs-Brock et al., 1990] Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wirth, 1971] Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

[X3J16/WG21, 1996] ANSI X3J16 and ISO WG21. "*Working Paper for Draft Proposed International Standard for Information Systems—Programming Language C++*". ftp://research.att.com/dist/c++std/WP/CD2. [Última vez visitado 7/1/2000]. X3J16/96–0225 (WG21/N1043). December, 1996.

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice-Hall, 1989.

[Zilles, 1984] Zilles, S. "*Types, Algebras, and Modeling*". In *On Conceptual Modeling: Perspectives from Artificial Inteligence, Databases, and Programming Languages*. Springer-Verlag, 1984.

Apéndice A
Plan de Calidad de la Unidad Docente de IS y OO

Se incluye íntegramente la versión 1.1 del Plan de Calidad de la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca para el bienio 1999-2000, cuya última modificación data del 20 de marzo de 2000.

Este plan de calidad ha sido elaborado por el *Dr. Francisco José García Peñalvo,* la *Dra. María N. Moreno García,* el *Dr. José Rafael García-Bermejo Giner* y *Ana de Luis Reboredo*.

A.1 Introducción

Hay muchas definiciones de calidad en la bibliografía, según la norma ISO 8402 (admitida como norma española UNE 66-001-92), la calidad se define como: "*Totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas*".

El concepto de calidad es un objetivo fundamental para los directivos de una empresa junto a los dos parámetros clásicos de su gestión: el dinero y el tiempo. El mercado actual es sumamente competitivo, donde no basta con producir masivamente los productos o servicios, vender es lo importante y sólo se produce un producto cuando se tiene la seguridad de aceptación por parte del cliente. Así, conseguir la satisfacción del cliente y conocer sus necesidades, para luego definirlas en forma de requisitos a cumplir, se convierte en una necesidad imperiosa para las empresas que pretendan hacerse con un hueco en un determinado mercado.

En la vida cotidiana, la calidad representa las propiedades inherentes a un objeto, de forma que pueda ser comparado con otros objetos de su especie para determinar si es mejor, igual o peor. Calidad es sinónimo de bondad, excelencia o superioridad.

Para la comunidad universitaria, la calidad debe ser un objetivo tan importante como para la empresa. El profesor de Universidad debe ver en la calidad un camino hacia la excelencia en todas sus actividades: docencia, investigación y gestión.

En lo referente a la actividad investigadora, la calidad debe ser la guía para aumentar el conocimiento del investigador, y debe estar presente en todos los productos resultantes de la investigación.

La actividad docente por su parte puede asociarse más a una actividad contractual donde la Universidad es la empresa y los clientes son varios. El cliente más directo es el alumno, el producto que se le ofrece debe ser un currículo que cumpla unos requisitos acordes a los objetivos marcados por la titulación elegida. La sociedad es el cliente indirecto de la Universidad, donde el producto que se le ofrece son los alumnos formados y preparados para introducirse en el mundo real y rendir acorde a las necesidades de esa sociedad. Por último, el propio docente será a la vez mecanismo y cliente de la Universidad, al buscar por un lado la excelencia de conocimiento en una determinada materia y por otro lado la satisfacción personal conseguida cuando los alumnos salen cumpliendo los requisitos de conocimientos que la Universidad marca y la adecuación que la sociedad demanda.

Entre los cursos académicos 96-97 y 98-99 se han desarrollado una serie de actividades destinadas a la mejora continua en el ámbito de la docencia en los campos de la Ingeniería del Software y de la Orientación a Objetos, las cuales se han visto plasmadas en un conjunto de planes de calidad que se centraban en recoger los objetivos y los criterios de evaluación de los mismos para asignaturas concretas, cayendo la responsabilidad de poner en marcha estos planes en la persona que acometía esta tarea [García et al., 1999b].

Como complemento a esta actividad de carácter meramente individual, se estableció un mecanismo de evaluación externa, que permitiese al responsable de la misma obtener una mayor información contrastada con la que poder realimentar el proceso de mejora continua [García et al., 1999c].

Sin embargo, estas iniciativas individuales no han influido de una forma clara en la obtención de una mejora global dentro de un colectivo (área, departamento, titulación...) al estar situadas en los límites de una sola asignatura.

Por este motivo, se va acometer la tarea de la realización de un plan de calidad para el bienio 1999-2001, pero no centrado en una única asignatura, sino que abarque una unidad docente completa, la unidad docente de Ingeniería del Software y Orientación a Objetos, que tiene competencias en dos titulaciones universitarias diferentes, aunque

relacionadas: la *Ingeniería Técnica en Informática de Sistemas (Plan de 1997)* y la *Ingeniería Informática*, ambas impartidas en la Universidad de Salamanca.

El plan de calidad se estructura como sigue: en la sección dos se presenta y se justifica la unidad docente de Ingeniería del Software y Orientación a Objetos, especificando los objetivos perseguidos por esta unidad docente, las asignaturas que la forman, indicando sus interrelaciones, y estableciendo a groso modo como se reparten los objetivos de la unidad docente en las asignaturas. La tercera sección se centra en detallar de forma concreta los objetivos y las aportaciones de cada una de las asignaturas, con la misión específica, en esta primera aproximación para el bienio 1999-2001, de minimizar solapamientos de contenidos y de conocer explícitamente los contenidos y limites del resto de las asignaturas que conforman la unidad docente.

A.2 Unidad docente de Ingeniería del Software y Orientación a Objetos

La Ingeniería del Software es la parte de la disciplina informática que se ocupa de ofrecer una aproximación sistemática a la creación y mantenimiento de los sistemas software, ofreciendo un enfoque que rehuye las concepciones tradicionales, por las que el software se concibe como un producto artesano inmerso en un ambiente *mitológico*, casi *místico*, donde los sistemas software distan mucho de las necesidades reales del tejido social y empresarial actual.

En la aproximación buscada por la Ingeniería del Software se concibe al *ingeniero informático* como aquella persona que es capaz de aplicar sus conocimientos para la resolución de los problemas reales derivados de la concepción, explotación y mantenimiento de sistemas informáticos, donde la perspectiva no es la de pequeños sistemas, que pueden ser responsabilidad de un único individuo, sino de los sistemas que requieren para su realización el trabajo conjunto y coordinado de diferentes personas.

En España, actualmente, los estudios universitarios de informática se organizan en ingenierías técnicas e ingenierías superiores, siendo una política acorde con los que defienden que la informática es una ingeniería [Buxton et al., 1976], [Shaw and Tomayko, 1991], [Leveson, 1997], [Parnas, 1997]. Así, el cuerpo de conocimientos de un ingeniero informático debe estar formado por una sólida base en Lógica, Matemáticas y Ciencias de la computación, aspectos propios de Ingeniería del Software y un conjunto de temas de carácter universal que completarán su currículo (economía, idiomas...).

La unidad docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca, se encarga

de impartir la docencia relacionada con la Ingeniería del Software en las titulaciones que se discuten.

La inclusión de la orientación a objetos como parte de la unidad docente se debe a la gran repercusión de los métodos orientados a objetos en el desarrollo de sistemas software, lo cual lleva a hablar de una Ingeniería del Software Orientada a Objetos que, por las propias características intrínsecas de la tecnología de objetos, abarca con un mayor énfasis todas las fases del ciclo de vida del software.

A.2.1 Objetivos de la unidad docente

Los objetivos de la unidad docente se pueden clasificar en tres grandes apartados:

- Conceptos teóricos
- Aspectos prácticos
- Habilidades personales

A.2.1.1 Conceptos teóricos

T1 Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.

T2 Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

T5 Especificaciones formales de requisitos.

T6 Método de análisis/diseño estructurado.

T7 Método de análisis/diseño orientado a objetos.

T8 Diseño de la interfaz gráfica de usuario.

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

T10 Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

T11 Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

T13 Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

A.2.1.2 Aspectos prácticos

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

P3 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

P5 Programación orientada a objetos.

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

P7 Aprendizaje y manejo de forma práctica de plataformas, entornos de desarrollo, lenguajes de programación... de alta repercusión en el desarrollo de sistemas software en la actualidad.

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

P9 Construcción de sistemas software de entidad superior a una práctica de laboratorio, a ser posible partiendo de unas especificaciones reales elicitadas a *clientes* y/o *usuarios* reales.

A.2.1.3 Habilidades personales

H1 Mejora de la expresión oral.

H2 Mejora en la redacción de documentos técnicos.

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

H4 Capacitar a los alumnos para el trabajo en grupo.

A.2.2 Asignaturas de la unidad docente

La unidad docente de Ingeniería del Software y Orientación a Objetos se ha creado para dar cobertura a las titulaciones de Ingeniería Técnica en Informática de Sistemas (I.T.I.S) (Plan de 1997) e Ingeniero en Informática (I.I) (Plan de 1998) impartidas en la Universidad de Salamanca.

La primera de ellas es una titulación de primer ciclo (de tres cursos de duración), mientras que la segunda es una titulación de segundo ciclo (de dos cursos de duración), que tiene como condición de acceso la posesión del título de Ingeniero Técnico en Informática de Gestión o Ingeniero Técnico en Informática de Sistemas.

En estas titulaciones las asignaturas que mejor se ajustan a los objetivos marcados en la unidad docente se recogen en la Tabla A.1.

Asignatura	Titulación	Curso	Carácter	Créditos
Interfaces Gráficas	I.T.I.S	2º	Optativa	6 (3T + 3P)
Ingeniería del Software	I.T.I.S	3º	Obligatoria	6 (4,5T + 1,5P)
Programación Orientada a Objetos	I.T.I.S	3º	Optativa	6 (3T + 3P)
Proyecto	I.T.I.S	3º	Obligatoria	9 (9P)
Análisis de Sistemas	I.I	1º	Troncal	9 (6T + 3P)
Administración de Proyectos Informáticos	I.I	2º	Troncal	9 (6T + 3P)
Sistemas de Información	I.I	2º	Troncal	9 (9P)
Proyecto	I.I	2º	Troncal	6 (6P)

Tabla A.1. Asignaturas que componen la unidad docente

Las dependencias e interrelaciones entre estas asignaturas se muestran en la Figura A.1. En el establecimiento de estas dependencias se ha tenido en cuenta tanto el factor tiempo, que claramente establece el orden lógico en el que se van a cursar las asignaturas, como las aportaciones a la unidad docente de las mismas en forma de contenidos y objetivos a conseguir.

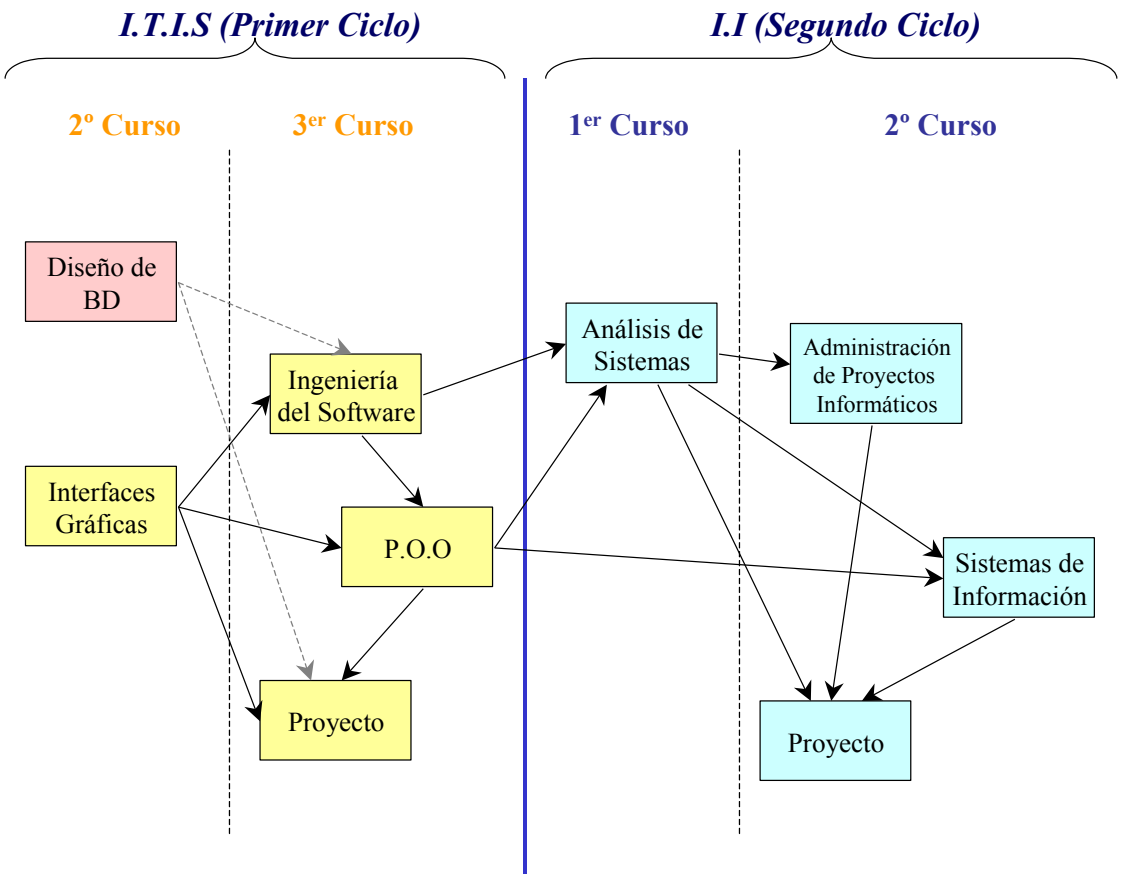

Figura A.1. Dependencias entre las asignaturas que forman la unidad docente

Además de las asignaturas propias de la unidad docente, se incluye la asignatura de **Diseño de Bases de Datos**. Esto se debe a que es la asignatura en la que se introducen los modelos de datos conceptuales y lógicos (diagramas entidad-interrelación y modelos relacionales típicamente), lo que supone una importante base, a la vez que una descarga, para la asignatura de Ingeniería del Software donde estos modelos serán utilizados de forma práctica sin necesidad de tener que incluirlos en la parte teórica de la asignatura.

A.2.3 Reparto de los objetivos en las asignaturas de la unidad docente

	Obj. Teóricos	Obj. Prácticos	Hab. Personales
Interfaces Gráficas	T8	P6	H1, H2, H3
Ingeniería del Software	T1, T2, T3, T4, T6, T7, T9	P1, P2, P4	H1, H2, H3, H4
Programación Orientada a Objetos	T7, T9	P2, P5	H1, H2, H3, H4
Proyecto I.T.I.S		P9	H1, H2, H3
Análisis de Sistemas	T3, T4, T5, T7, T12, T13	P1, P2, P4	H1, H2, H3, H4
Administración de Proyectos Informáticos	T10, T11, T12	P4, P8	H1, H2, H3, H4
Sistemas de Información		P7	H1, H2, H3
Proyecto I.I		P8, P9	H1, H2, H3

Tabla A. 2. Reparto de los requisitos en las asignaturas de la unidad docente

A.3 Estudio de los objetivos de cada una de las asignaturas

A.3.1 Interfaces gráficas

El objetivo de esta asignatura es introducir a los alumnos de I.T.I.S en algunos sistemas de desarrollo de interfaces gráficas de frecuente uso en la actualidad. Se trata pues de un objetivo fundamentalmente práctico, ya que se busca la capacitación del alumno para implementar aplicaciones que incorporen los elementos habituales en las interfaces gráficas de usuario actuales.

Este objetivo se corresponde con los objetivos **T8, P6, H1, H2** y **H3** de la unidad docente.

A continuación, se detallan los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.1.1 Ficha de la asignatura

Asignatura	*Interfaces gráficas (Optativa)*
Créditos	*3T + 3P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*2º*
Cuatrimestre	*2º*
Responsable	*Ana de Luis Reboredo (adeluis@gugu.usal.es)*
Página web de la asignatura	

Tabla A.3. Interfaces gráficas

A.3.1.2 Prerrequisitos

Es imprescindible que el alumno esté capacitado para el desarrollo de programas en lenguaje C. Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

Sería muy conveniente que el alumno ya dispusiese de ciertos conocimientos de programación orientada a objetos, que le facilitaran el acceso a ciertas herramientas de creación de interfaces gráficas actuales. No obstante, con la actual distribución temporal de asignaturas, esto no es muy frecuente. Por ello, en la asignatura se proporcionarán unos conocimientos básicos de programación orientada a objetos y, en concreto, del lenguaje C++ que permitan a los alumnos utilizar dichas herramientas.

A.3.1.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: La GUI de Windows. Programación SDK (11 Horas)
Tema 1. Introducción a la programación en Windows (1 Hora)
Tema 2. GDI. Fundamentos gráficos (2 Horas)
Tema 3. Eventos de entrada. Ratón y teclado (1 Hora)
Tema 4. Controles. Controles predefinidos (2 Horas)
Tema 5. Recursos (1 Hora)
Tema 6. Menús (1 Hora)
Tema 7. Cuadros de diálogo (2 Horas)
Tema 8. Intercambio de datos (1 Hora)
Unidad Docente II: Visual C++ (8 Horas)
Tema 8. Introducción a la programación orientada a objetos (2 Horas)
Tema 9. La biblioteca MFC (6 Horas)
Unidad Docente III: El entorno X Window (10 Horas)
Tema 10. Introducción. Arquitectura del sistema (2 Horas)
Tema 11. Conceptos básicos. Terminología (2 Horas)
Tema 12. Ventanas (2 Horas)
Tema 13. Entradas. El ratón y el teclado (2 Horas)
Tema 14. Gráficos (2 Horas)

Tabla A.4. Programa de teoría de Interfaces Gráficas (3 créditos)

Prácticas (30 Horas)

Realización de aplicaciones mediante programación SDK (12 Horas)
Realización de aplicaciones con Visual C++ (8 Horas)
Realización de aplicaciones con X Window (10 Horas)

Práctica obligatoria

Realización de una aplicación completa que incorpore la mayoría de los elementos estudiados.

Tabla A.5. Programa de prácticas de Interfaces Gráficas (3 créditos)

A.3.1.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir.

T8 Diseño de la interfaz gráfica de usuario.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación de los diferentes elementos que componen una GUI y sus distintos modos de implementación	Clases Teóricas	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de sucesivas aplicaciones que incorporen elementos de interfaces gráficas para Windows y X Window	Clases prácticas	Responsable de la asignatura Alumnos	Aula de Informática Transparencias Bibliografía básica	Defensa de la práctica obligatoria Examen

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Ordenador	Defensa de la práctica

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Documentación de la práctica obligatoria	Defensa de la práctica	Grupo de alumnos Responsable de la asignatura	Bibliografía	Defensa de la práctica

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca Recursos de Internet	Examen

A.3.1.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (*50% de la nota final*).

- Un examen final en el mes de Junio.

- Un examen final en el mes de Septiembre.

b) Parte Práctica (*50% de la nota final*)

- Desarrollo de una aplicación con una interfaz gráfica de usuario que incluya la mayoría de los elementos estudiados. Se realizará en grupos de un máximo de tres miembros.

Se realizará una defensa por grupos de dicho trabajo.

Tras la defensa, los diferentes miembros del grupo deberán responder a las preguntas que se les planteen de forma individualizada sobre el desarrollo de la aplicación y las posibles modificaciones de la misma.

La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Cada una de las partes se deberá aprobar por separado exigiéndose un mínimo de 5 puntos en cada una.

A.3.1.6 Bibliografía básica de referencia

🕮 **Petzold, C.** "*Programación en Windows 95*". McGraw-Hill, 1996.

Es la traducción del libro "*Programming Windows 95*" que a su vez es la continuación del clásico "*Programming Windows 3.0*" del mismo autor. Es el libro básico para la introducción en la programación SDK, imprescindible para comprender la programación en Windows. Explica detalladamente el funcionamiento de la API de Windows desarrollando de modo progresivo los distintos aspectos relacionados con la misma. Cuenta con multitud de ejemplos en C que están disponibles en CD. Se utilizará para todos los temas de la Unidad Docente I de la asignatura.

🕮 **Reiss, L. y Radin, J.** "*Aplique X Window*". McGraw-Hill, 1993.

Se trata de la traducción del libro de los mismos autores "*X Window Inside & Out*". Parte de una completa introducción al sistema y sus componentes para pasar a describir las funciones X Window ilustrándolas con programas completamente desarrollados. Será el libro que se utilizará en la Unidad Docente III de la asignatura.

🕮 **Yao, P. y Leinecker, R.C.** "*Todo Visual C++ 5* ". INFORBOOK'S, 1998.

Es la traducción del libro "*Visual C++ 5 Bible*" de los mismos autores. El contenido está dividido en 4 partes. La primera parte contiene una introducción a la programación en Windows en la que hace algunas consideraciones sobre la programación con la API de Microsoft Windows. A continuación, incluye una pequeña guía de uso de la herramienta de desarrollo Visual C++. La segunda parte es una introducción a la programación orientada a objetos y a las ampliaciones del lenguaje C++ sobre el lenguaje C. La tercera parte trata todos los aspectos relacionados con la librería MFC. La última parte incluye algunos temas de interés como, por ejemplo algunos formatos de archivos de imágenes o la gestión de memoria de Windows. Todo el libro está ilustrado con abundantes ejemplos que están disponibles en un CD adjunto.

A.3.2 Ingeniería del Software

El objetivo principal de esta asignatura es introducir a los alumnos de I.T.I.S en los principios fundamentales, métodos y herramientas para el desarrollo sistemático de proyectos informáticos. Adicionalmente, los alumnos deberán aplicar éstas trabajando en equipo.

Este objetivo se corresponde con los objetivos **T1, T2, T3, T4, T6, T7, T9, P1, P2, P4, H1, H2, H3** y **H4** de la unidad docente.

Para ver satisfacer estos objetivos, se va a detallar los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.2.1 Ficha de la asignatura

Asignatura	*Ingeniería del software (obligatoria)*
Créditos	*4,5T + 1,5P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*1º*
Responsable	*Francisco José García Peñalvo (fgarcia@gugu.usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla A.6. Ingeniería del Software

A.3.2.2 Prerrequisitos

El alumno debe estar familiarizado con la teoría así como con la práctica del diseño y codificación en lenguajes procedurales (por ejemplo C). Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

El alumno debe tener los conocimientos y la práctica necesaria en la creación de modelos de información conceptuales y lógicos, en concreto, dominio del modelo entidad-relación, paso al modelo relacional y normalización. Estos conceptos se adquieren en la asignatura de segundo curso **Diseño de Bases de Datos** (*asignatura troncal en el plan de estudios vigente*).

Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

A.3.2.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (6 Horas)
Tema 1. Introducción a la Ingeniería del Software (6 Horas)
Unidad Docente II: Paradigma estructurado de desarrollo (23 Horas)
Tema 2. Análisis y especificación de requisitos (11 Horas)
Tema 3. Análisis estructurado (2 Horas)
Tema 4. Diseño del software (6 Horas)
Tema 5. Diseño estructurado (4 Horas)
Unidad Docente III: Introducción al paradigma objetual (13 Horas)
Tema 6. Introducción a la Orientación a Objetos (6 Horas)
Tema 7. UML (6 Horas)
Tema 8. Visión general de la metodología OMT (1 Hora)
Unidad Docente IV: Miscelánea (2 Horas)
Tema 9. Herramientas CASE (2 Horas)

Tabla A.7. Programa teórico de Ingeniería del Software (4,5 créditos)

Talleres (10 Horas)
Práctica 1. Taller de modelado de datos. Repaso al modelo entidad-interrelación (2 Horas)
Práctica 2. Taller de modelado funcional (Enfoque Clásico)(2 Horas)
Práctica 3. Taller de modelado funcional (Yourdon) (4 Horas)
Práctica 4. Taller de orientación al objeto (2 Horas)
Laboratorio (2 Horas)
Práctica 5. Manejo de una herramienta CASE (2 Horas)
Práctica obligatoria
Práctica 6. Realización de una ERS y un prototipo de una aplicación software

Tabla A.8. Programa práctico de Ingeniería del Software (1,5 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.9 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T1, T2, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T9, H3
Tema 5	T6, H3
Tema 6	T3, T7, H3
Tema 7	T7, H3
Tema 8	T7, H3
Tema 9	P4, H3
Práctica 1	P1, H1, H2, H4
Práctica 2	P1, H1, H2, H4
Práctica 3	P1, H1, H2, H4
Práctica 4	P2, H1, H2, H4
Práctica 5	P4
Práctica 6	P1, P2, P4, H1, H2, H4

Tabla A.9. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

La Unidad Docente IV: Miscelánea, es difícil de impartir en la realidad por limitaciones de tiempo. El Tema 9, dedicado a las herramientas CASE, puede introducirse a la vez que se realiza la Práctica 4.

A.3.2.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir. Cada línea de acción va a definirse mediante los cuatro apartados indicados en [García et al., 1999b]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T1 Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.

T2 Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación de la problemática del desarrollo de programas con calidad industrial, poniendo de manifiesto la necesidad de emplear técnicas de ingeniería	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Reuniones con los alumnos Examen
Presentación de ejemplos de empresas que hagan uso de métodos de Ingeniería del Software bien implantados	A lo largo de la asignatura Conferencias impartidas por parte de representantes del mundo empresarial	Ponentes invitados	Medios audiovisuales Contactos en la empresa	Opinión de los alumnos Opinión de otros profesores del Departamento
Estudio y comprensión de los diferentes paradigmas de ciclo de vida de un desarrollo software	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación del papel protagonista de los requisitos y de las especificaciones dentro de un desarrollo software	Durante toda la asignatura, especialmente en la Unidad II, Unidad III y en la práctica de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Reuniones con los alumnos Examen
Distinción entre elicitación, documentación y especificación de requisitos	Unidad docente II. Se habla de requisitos de cliente y de desarrollador	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T6 Método de análisis/diseño estructurado.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas y modelos para la especificación de requisitos y diseño en el paradigma estructurado	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Introducción de las extensiones de notación funcional para tiempo real	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método estructurado para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T7 Método de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas y modelos para la especificación de requisitos y diseño en el paradigma objetual	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Ver los fundamentos de diseño como la base necesaria para obtener sistemas software de calidad, con independencia del paradigma de desarrollo empleado	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de talleres en parte de las horas prácticas de la asignatura donde los alumnos se organizan en grupos para resolver un problema utilizando técnicas estructuradas y orientadas a objeto. Posteriormente uno de los grupos expondrá su solución y se debatirá sobre ella	Parte práctica de la asignatura	Grupos de alumnos Responsable de la asignatura	Enunciados con casos prácticos Transparencias	Informes realizados por los alumnos Comentarios de los alumnos
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Introducción a las herramientas CASE	Unidad docente IV Clase práctica	Responsable de la asignatura Alumnos	Transparencias Bibliografía Herramientas CASE	Defensa de la práctica
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Promover los debates sobre las soluciones presentadas en los talleres prácticos	Clases prácticas	Responsable de la asignatura Grupos de alumnos	Pizarra Medios audiovisuales	Estudio del seguimiento de los talleres Opinión alumnos
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para esos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de un informe técnico por cada uno de los talleres prácticos realizados. Estos informes quedarán publicados en la página web de la asignatura	En las 3 semanas siguientes a una clase práctica	Responsable de la asignatura Grupos de alumnos	Página web de la asignatura	Nota *extra* para estos trabajos voluntarios
Documentación de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para esos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
La práctica obligatoria debe ser realizada en grupos de trabajo de 3 a 5 personas	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados pos los alumnos	Defensa de la práctica
Realización de talleres en parte de las horas prácticas de la asignatura donde los alumnos se organizan en grupos para resolver un problema utilizando técnicas estructuradas y orientadas a objeto. Posteriormente uno de los grupos expondrá su solución y se debatirá sobre ella	Parte práctica de la asignatura	Grupos de alumnos Responsable de la asignatura	Enunciados con casos prácticos Transparencias	Informes realizados por los alumnos Comentarios de los alumnos

A.3.2.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (*50% de la nota final*)

- Un examen final en el mes de febrero.

- Un examen final en el mes de septiembre.

b) Parte Práctica (*50% de la nota final*)

- ERS y prototipo realizado en grupos de trabajo.

Se realizará una defensa individualizada de dicho trabajo.

La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Entre 0,5 y 0,75 puntos sumados a la nota de la parte práctica por la defensa y posterior redacción de un informe sobre los supuestos tratados en los talleres prácticos.

d) Trabajos voluntarios presentados (de *0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida*)

e) Fórmula para la obtención de la calificación final.

Si (Teoría ≥ 4,75) **y** (Práctica ≥ 5.0)

Nota Final = (Teoría*0,5) + ((Práctica+Nota Talleres)*0,5) + Nota trabajos

Sino

Ø

Fin si

A.3.2.6 Bibliografía básica de referencia

En este apartado se va a incluir aquellas referencias que el alumno debe consultar para completar el material que el responsable facilita para seguir las clases (transparencias). Estos títulos han sido elegidos en función de tres factores: *su adecuación a los contenidos, su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

Existen otras referencias que se han manejado por el responsable de la asignatura para preparar los temas, pero éstas, al ser más específicas, no son incluidas en este apartado, aunque aparecen citadas en las transparencias y son recomendadas como otras lecturas en muchas ocasiones.

🕮 **Booch, Grady, Rumbaugh, James and Jacobson, Ivar.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

Este libro es la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. De la trilogía de libros publicados por los tres máximos responsables del lenguaje modelado UML, éste es el que se corresponde con la explicación detallada de la notación de este lenguaje de modelado. Es un libro de consulta obligada para el tema 6, aunque su primera parte de introducción, especialmente el capítulo 1, se puede utilizar en el primer tema de la parte teórica.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Corresponde a la traducción del libro "Object-Oriented Software Construction", 2nd Edition, Prentice-Hall, 1997. Esta segunda edición es más que una simple actualización de la primera edición, con una ampliación de conceptos más que considerable. Aunque es un libro que se adecua más a una asignatura de programación orientada a objetos, los contenidos de los tres primeros capítulos se adaptan muy bien a la asignatura de Ingeniería del Software, especialmente el Capítulo 3 dedicado a la modularidad, que es una referencia importante para el Tema 4.

🕮 **Piattini Velthuis, M. G., Calvo-Manzano, J. A., Cervera, J. y Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Libro de introducción a la Ingeniería del Software, centrado en la construcción de software de gestión. Consta de tres partes: Sistemas de Información (capítulos 1 y 2), El Proceso de Desarrollo de Software (capítulos 3-16) y Tecnología (capítulos 17-19).

Es un libro de consulta que se puede utilizar en todos los temas de la parte teórica de la asignatura, aunque con una mayor profusión en los temas 1, 2, 4 y 5.

🕮 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill, 1998.

Es uno de los mejores libros de referencia y consulta para la práctica totalidad de los temas relacionados con la Ingeniería del Software. Corresponde a la traducción del libro del mismo autor "*Software Engineering: A Practitioner´s Approach*", 4th Edition, McGraw-Hill, 1997. En este texto se realiza una amplia exposición de prácticamente todos los temas relacionados con la Ingeniería del Software desde la doble perspectiva del producto y el proceso. Es de destacar el conjunto de referencias tanto bibliográficas como de recursos disponibles en Internet de cada uno de los temas tratados. Se considera un texto imprescindible de referencia para esta asignatura. En este sentido hay que señalar que en el prólogo de esta edición española se indica que explica todos los temas incluidos en la asignatura o materia troncal Ingeniería del Software de los planes de estudios de las universidades españolas.

En lo que respecta a la asignatura, se adecua como una referencia especialmente interesante en los temas 1, 2, 4, 5, 7 y 8.

🕮 **Rational Software Corporation.** "*The Unified Modeling Languague Documentation Set 1.1*". http://www.rational.com. September, 1997.

Esta documentación corresponde al texto completo de la propuesta de UML realizada por Rational al OMG. Esta documentación consta de un metamodelo formal, una notación y su semántica. En el metamodelo se realiza una exposición de los conceptos e ideas, de la sintaxis y de la semántica de UML. En la notación se describen los diferentes tipos de diagramas: de casos de uso, de clases, de secuencias, de colaboración, de estados, de actividad, de componentes y de despliegue. En el documento de semántica se establece la semántica de los elementos de UML utilizando un subconjunto de la notación. Esta documentación es básica para el seguimiento de la Unidad Didáctica III de la asignatura.

🕮 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Corresponde a la traducción del libro de los mismos autores "*Object Oriented Modeling and Design*", Prentice Hall, 1991. En el libro se presenta una de las primeras metodologías orientadas al objeto que fueron desarrolladas, OMT. En esta metodología de desarrollo de sistemas se considera que el modelado orientado al objeto consta de tres partes: el modelo objeto, el modelo funcional y el modelo dinámico. A pesar de ser una metodología superada en muchos aspectos, desde el punto de vista de la docencia a impartir tiene como ventaja que toma algunas de las herramientas de las metodologías clásicas, por ejemplo el DFD para el elaborar el modelo funcional, lo que permite realizar una transición suave desde los métodos clásicos a los de la Orientación al Objeto. La tercera parte del libro aborda los aspectos de paso del diseño a la implementación en relación con los lenguajes de programación y las bases de datos relacionales. Incluye la descripción de tres casos de estudio. Este libro se utiliza principalmente como referencia para los temas 1 y 6.

🕮 **Sommerville, Ian**. "*Software Engineering*". 5th Edition, Addison-Wesley. 1996.

Esta es la última edición de uno de los libros más difundidos sobre Ingeniería del Software. Explica un amplio conjunto de técnicas y muestra como pueden ser aplicadas en la práctica de la Ingeniería del Software. Los temas están agrupados de acuerdo a las actividades del proceso de desarrollo y mantenimiento del software. Respecto a las versiones anteriores hay que destacar la incorporación de temas como re-ingeniería o la ampliación de temas como CASE y gestión y evolución del software. El autor, profesor de la asignatura de Ingeniería del Software en una universidad inglesa, se ha preocupado de adaptar los contenidos a la recomendación curricular ACM/IEEE-CS91 para los módulos SE2 a SE5. Además, puede obtenerse material suplementario de apoyo a la docencia en Internet. El nivel de algunos de los capítulos es superior al exigido a los alumnos de I.T.I.S, pero se considera uno de los libros imprescindibles de consulta en la materia.

🕮 **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

Corresponde a la traducción del texto del mismo autor "*Modern Structured Analysis*", Prentice-Hall, 1989. En él, Yourdon revisa las teorías del análisis estructurado desarrolladas por DeMarco en el año 1979. Es un libro con un diseño muy didáctico y adaptado a un curso de análisis estructurado. Tiene algunos capítulos, que en su momento constituyeron un cierto avance en las metodologías estructuradas y que, a pesar del paso de los años, mantienen su vigencia. Es de destacar el capítulo en el que se realiza un estudio general de sistemas, la propuesta de los diferentes modelos a realizar de un sistema y los casos prácticos desarrollados en los apéndices. Sigue constituyendo uno de los libros de referencia obligada para los cursos de Ingeniería del Software. En esta asignatura es una referencia complementaria en los temas 1 y 2, siendo la referencia fundamental del tema 3.

A.3.3 Programación Orientada a Objetos

El principal objetivo de esta asignatura es transmitir al alumno los conocimientos necesarios para el diseño e implementación de aplicaciones software bajo el paradigma objetual.

Aunque se va a utilizar un lenguaje orientado a objetos (C++) para que el alumno pueda llevar al terreno práctico los conceptos transmitidos en esta asignatura, se pretende una exposición de los mismos desde un plano más abstracto, más centrado en el diseño, de forma que el alumno pueda aplicarlos en su vida laboral con independencia del entorno de desarrollo orientado a objetos con el que se encuentre.

Este objetivo general se corresponde con los objetivos concretos **T7, T9, P2, P5, H1, H2, H3** y **H4** de la unidad docente.

Para ver satisfacer dichos objetivos, se va a detallar los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.3.1 Ficha de la asignatura

Esta asignatura se considera una extensión de la asignatura Ingeniería del Software, en la que se amplía la Unidad Docente III de ésta con aspectos de diseño y programación orientada objetos. De hecho, la asignatura de Programación Orientada a Objetos se imparte en el segundo cuatrimestre del tercer curso de I.T.I.S, después de la finalización de la asignatura de Ingeniería del Software.

Asignatura	*Programación orientada a objetos (optativa)*
Créditos	*3T + 3P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*2º*
Responsable	*Francisco José García Peñalvo (fgarcia@gugu.usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla A.10. Programación Orientada a Objetos

A.3.3.2 Prerrequisitos

El alumno debe estar familiarizado con la teoría así como con la práctica del diseño y codificación en lenguajes procedurales (por ejemplo C). Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

El alumno debe conocer los fundamentos básicos que rigen un proyecto software, estar familiarizado con la elicitación y especificación de requisitos y manejar los principios de la Ingeniería del Software orientada a objetos, en concreto el lenguaje de modelado UML y conocimientos generales de una metodología orientada a objetos del tipo OMT. Todos estos conceptos le son transmitidos al alumno en la asignatura de tercer curso **Ingeniería del Software**.

A.3.3.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (4 Horas)
Tema 1. Lenguajes de programación orientados a objetos (2 Horas)
Tema 2. Orientación a objeto y reutilización del software (2 Horas)
Unidad Docente II: Diseño orientado a objetos básico (14 Horas)
Tema 3.Técnicas básicas de diseño orientado a objetos (8 Horas)
Tema 4. Genericidad (4 Horas)
Tema 5. Manejo de excepciones (2 Horas)
Unidad Docente III: Diseño orientado a objetos avanzado (11 Horas)
Tema 6. Patrones de diseño (8 Horas)
Tema 7. Diseño por contrato (3 Horas)

Tabla A.11. Programa de teoría de Programación Orientada a Objetos (3 créditos)

C++ (18 Horas)
Tema C++ 1. Modelo objeto en C++ (10 Horas)
Tema C++ 2. STL (8 Horas)
Taller (2 Horas)
Tarjetas CRC
Prácticas libres (10 Horas)
Práctica obligatoria
Realización de una aplicación utilizando técnicas orientadas a objeto

Tabla A.12. Programa de prácticas de Programación Orientada a Objetos (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.13 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T7
Tema 2	T7
Tema 3	T7, T9, P2
Tema 4	T7, T9, P2
Tema 5	T7, T9, P2
Tema 6	T7, T9, P2
Tema 7	T7, T9, P2
Tema C++ 1	P5
Tema C++ 2	P5
Taller CRC	P2, H1, H2, H3, H4
Práctica obligatoria	P2, P5, H1, H2, H3, H4

Tabla A.13. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.3.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir. Cada línea de acción va a definirse mediante los cuatro apartados indicados en [García et al., 1999b]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T7 Método de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Desarrollo de aplicaciones bajo el paradigma objetual, centrando la atención en las fases de diseño e implementación	Clases teóricas y prácticas	Responsable de la asignatura	Bibliografía Transparencias Lenguaje de programación orientado a objetos	Examen teórico Prácticas Opinión de los alumnos

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Aplicación de principios y técnicas del diseño orientado a objetos	Parte teórica	Responsable de la asignatura	Transparencias Bibliografía básica	Examen Práctica obligatoria
Introducción de técnicas avanzadas de diseño orientado a objetos	Parte teórica	Responsable de la asignatura	Transparencias Bibliografía básica	Examen Práctica obligatoria

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de una práctica obligatoria en la que se cree una aplicación utilizando técnicas orientadas a objeto	Parte práctica de la asignatura	Grupos de alumnos	Caso práctico	Defensa de la práctica

P5 Programación orientada a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de un lenguaje orientado a objetos	Clases prácticas	Responsable de la asignatura Alumnos	Bibliografía Pequeños supuestos prácticos	Defensa de la práctica
Realización de una práctica obligatoria en la que se cree una aplicación utilizando técnicas orientadas a objeto	Parte práctica de la asignatura	Grupos de alumnos	Caso práctico	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Documentación de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
La práctica obligatoria debe ser realizada en grupos de trabajo de 2 ó 3 personas	Práctica obligatoria de la asignatura	Alumnos Responsable de la asignatura	Supuestos prácticos	Defensa de la práctica

A.3.3.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (*50% de la nota final*)

- Un examen final.

b) Parte Práctica (*50% de la nota final*)

- Realización de un supuesto práctico por parejas.

  Se realizará una defensa de dicho trabajo.

  La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Trabajos voluntarios presentados (de *0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida*)

d) Fórmula para la obtención de la calificación final.

Si (Teoría ≥ 4,75) **y** (Práctica ≥ 5.0)

Nota Final = (Teoría*0,5) + (Práctica*0,5) + Nota trabajos

Sino

ø

Fin si

A.3.3.6 Bibliografía básica de referencia

En este apartado se va a incluir aquellas referencias que el alumno debe consultar para completar el material que el responsable facilita para seguir las clases (transparencias). Estos títulos han sido elegidos en función de tres factores: *su adecuación a los contenidos, su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

Existen otras referencias que se han manejado por el responsable de la asignatura para preparar los temas, pero éstas, al ser más específicas, no son incluidas en este apartado, aunque aparecen citadas en las transparencias y son recomendadas como otras lecturas en muchas ocasiones.

🕮 **Booch, Grady, Rumbaugh, James and Jacobson, Ivar.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

Este libro es la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. De la trilogía de libros publicados por los tres máximos responsables del lenguaje modelado UML, éste es el que se corresponde con la explicación detallada de la notación de este lenguaje de modelado.

Es un libro de consulta para repasar o aclarar cualquier duda que pudiera surgir sobre UML a lo largo del desarrollo de la asignatura.

🕮 **Deitel, H. M. and Deitel, P. J.** "*Como Programar en C/C++*". 2ª Ed. Prentice Hall, 1995.

Traducción del libro de los mismos autores "*C How to Program*", 2nd Edition, Prentice may, 1994. Es un libro con dos partes bien diferenciadas, los primeros catorce capítulos están dedicados al lenguaje C, mientras que los siete últimos a C++. Es por tanto un libro de referencia general que, además de tratar temas de C++, sirve como consulta para posibles dudas sobre lenguaje C.

🕮 **Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John.** "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

Este libro es un clásico del diseño orientado a objeto y el máximo responsable de la difusión de los patrones de diseño en la comunidad de la Orientación a Objetos. El libro es un catálogo de 23 patrones de diseño divididos en tres categorías: patrones de creación, patrones estructurales y patrones de comportamiento.

Es la referencia más importantes de la parte teórica de la asignatura, así como el texto base del tema 5.

🕮 **Glass, Graham and Schuchert, Brett**. "*The STL <Primer>*". Prentice-Hall, 1996.

Libro que hace un recorrido muy detallado por la biblioteca de plantillas estándar de C++ **STL** (*Standard Template Library*). El libro está dividido en cinco secciones: *Presentación de STL, Utilizando STL, Catálogo de las clases de STL, Catálogo de Algoritmos* y *Apéndices.*

Este libro sirve como referencia complementaria en el tema 3 de la parte teórica dedicado a la genericidad, y como referencia base en el tema 2 de prácticas dedicado a la biblioteca STL.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Corresponde a la traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997. Esta segunda edición es más que una simple actualización de la primera edición, con una ampliación de conceptos más que considerable. Es un libro sobre el cual se podía construir una asignatura de programación orientada a objetos, especialmente si elige Eiffel como lenguaje de programación para la parte práctica.

En la asignatura de Programación Orientada a Objetos va a ser una referencia de consulta más que una guía de los contenidos, aunque es la referencia clave en el tema 6.

🕮 **Rational Software Corporation.** "*The Unified Modeling Languague Documentation Set 1.1*". http://www.rational.com. September, 1997.

Esta documentación corresponde al texto completo de la propuesta de UML realizada por Rational al OMG. Esta documentación consta de un metamodelo formal, una notación y su semántica. En el metamodelo se realiza una exposición de los conceptos e ideas, de la sintaxis y de la semántica de UML. En la notación se describen los diferentes tipos de diagramas: de casos de uso, de clases, de secuencias, de colaboración, de estados, de actividad, de componentes y de despliegue. En el documento de semántica se establece la semántica de los elementos de UML utilizando un subconjunto de la notación.

Aunque el estudio de UML no es un objetivo de esta asignatura, sino de la asignatura de Ingeniería del Software, va a ser el lenguaje de modelado que se utilice para representar los diseños a discutir en la asignatura.

🕮 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.

Corresponde a la traducción del libro de los mismos autores "*Object Oriented Modeling and Design*", Prentice Hall, 1991. En el libro se presenta una de las primeras metodologías orientadas al objeto que fueron desarrolladas, OMT. En esta metodología de desarrollo de sistemas se considera que el modelado orientado al objeto consta de tres partes: el modelo objeto, el modelo funcional y el modelo dinámico. A pesar de ser una metodología superada en muchos aspectos, desde el punto de vista de la docencia a impartir tiene como ventaja que toma algunas de las herramientas de las metodologías clásicas, por ejemplo el DFD para el elaborar el modelo funcional, lo que permite realizar una transición suave desde los métodos clásicos a los de la Orientación al Objeto. La tercera parte del libro aborda los aspectos de paso del diseño a la implementación en relación con los lenguajes de programación y las bases de datos relacionales. Incluye la descripción de tres casos de estudio.

Este libro constituye una referencia de consulta válida en todos los temas de la parte teórica de la asignatura, especialmente por su excelente exposición del modelo objeto de OMT.

🕮 **Stroustrup, Bjarne.** "*El Lenguaje de Programación C++*". 3ª Ed. Addison-Wesley. 1998.

Este libro es la traducción de la obra del mismo autor "*The C++ Programming Language*", 3rd Edition, Addison-Wesley, 1997. Es el libro de referencia por excelencia cuando de C++ se trata, al ser el autor del mismo el padre de este lenguaje de programación orientado a objetos. En el momento de escribir esta tercera edición el proceso de estandarización de C++ se encontraba en su recta final, no previéndose cambios significativos, por lo que se puede considerar que el libro contiene una presentación de C++ actualizada y validada por el comité de estandarización de este lenguaje. El libro se divide en 6 partes: *Introducción* (capítulos 1-3), *Tipos Predefinidos en C++* (capítulos 4-9), *Programación Orientada a Objetos y Genérica con C++* (capítulos 10-15), *Biblioteca C++ Estándar* (capítulos 16-22), *Diseño y Desarrollo de Software* (capítulos 23-25) y *Apéndices*.

Es un libro que se ajusta perfectamente a la práctica de la asignatura, en especial las tres primeras partes.

A.3.4 Proyecto I.T.I.S

La realización del proyecto de final de carrera se convierte en una de las primeras oportunidades con las que cuenta un estudiante de una Ingeniería Técnica en Informática para aplicar de forma integrada los conocimientos teóricos y prácticos obtenidos en el conjunto de asignaturas cursadas en la carrera.

El objetivo principal de éste casa con los objetivos **P9, H1, H2** y **H3** de la unidad docente (*en los casos en que el proyecto se lleve a cabo por más de una persona tendría sentido incluir el objetivo **H4***).

En un proyecto de final de carrera el cometido y la bibliografía será establecida por el tutor o tutores del mismo, aunque como guía de referencia general para la realización y documentación de los proyectos final de carrera se tiene [García et al., 1999a], guía que se ajusta especialmente a los proyectos que tienen como cometido la creación de un sistema software.

A.3.5 Análisis de Sistemas

El propósito fundamental de esta asignatura es que los alumnos del segundo ciclo de Ingeniería en Informática profundicen en los conocimientos adquiridos en la asignatura *Ingeniería del Software* de primer ciclo, especialmente en lo que se refiere a métodos de análisis y especificación de requisitos de sistemas orientados a objetos y a técnicas formales de especificación.

Los objetivos de la unidad docente en que se desglosa son los siguientes:

T3, T4, T5, T7, T12, T13, P1, P2, P4, H1, H2, H3 y **H4**.

Seguidamente se expondrán los prerrequisitos de esta asignatura, necesarios para conseguir los objetivos mencionados, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta que se recomienda a los alumnos.

A.3.5.1 Ficha de la asignatura

Asignatura	*Análisis de Sistemas (troncal)*
Créditos	*6T + 3P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*1º*
Cuatrimestre	*1º y 2º (anual)*
Responsable	*María N. Moreno García (mmg@gugu.usal.es)*
Página web de la asignatura	*http://olivo.usal.es/~mmoreno/analisis.html*

Tabla A.14. Análisis de Sistemas

A.3.5.2 Prerrequisitos

Además de los prerrequisitos que se establecen en la asignatura Ingeniería del Software, de esta unidad docente, se deben tener conocimientos teóricos y prácticos de los objetivos de la Ingeniería del Software, de los modelos de ciclo de vida del software más utilizados y de los métodos de análisis y diseño estructurado. Asimismo sería conveniente conocer las bases del desarrollo de software orientado a objetos.

Las asignaturas que tratan estos aspectos son **Ingeniería del Software y Programación Orientada a Objetos**, que se imparten en 3º de ITIS.

A.3.5.3 Temario teórico/práctico

Presentación de la asignatura (1 hora)
Unidad Docente I: Definición del proceso de software (7 horas)
Tema 1. Modelos avanzados de ciclo de vida (3 Horas)
Tema 2. Prototipos (4 Horas)
Unidad Docente II: Métodos de desarrollo (36 horas)
Tema 3. Métodos estructurados. Técnicas avanzadas (6 Horas)
Tema 4. Métodos orientados a objetos (10 Horas)
Tema 5. El lenguaje UML y el proceso unificado (10 Horas)
Tema 6. Técnicas formales de especificación (10 Horas)
Unidad Docente III: Desarrollo de sistemas especiales (12 horas)
Tema 7. Sistemas de tiempo real (6 horas)
Tema 8. Otros sistemas (6 horas)
Unidad Docente IV: Evolución del software (4 horas)
Tema 9. Evolución y mantenimiento del software (4 horas)

Tabla A.15. Programa de teoría de Análisis de Sistemas (6 créditos)

Laboratorio: uso de herramientas CASE (30 horas)
Práctica 1. Análisis y Diseño estructurado de un sistema (15 horas)
Práctica 2. Análisis y Diseño orientado a objetos (15 horas)

Tabla A.16. Programa de prácticas de Análisis de Sistemas (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.17 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T3,T4, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T7, H3
Tema 5	T7, H3
Tema 6	T5, H3
Tema 7	T13, H3
Tema 8	T13, H3
Tema 9	T12, H3
Práctica 1	P1, H1, H2, H4
Práctica 2	P2, H1, H2, H4

Tabla A.17. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.5.4 Líneas de acción

En este apartado se comentarán las líneas de acción a seguir para la consecución de cada uno de los objetivos de la unidad docente. Cada línea de acción va a definirse mediante los cuatro apartados indicados en [García et al., 1999b]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Mostrar la relevancia de los requisitos al estudiar modelos avanzados de ciclo de vida.	En el tema 1 y en las prácticas de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Estudiar el prototipado de sistemas como modelo de ciclo de vida y como técnica de verificación y validación de requisitos	En el tema 2 y en las prácticas de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Realización del análisis y especificación de requisitos de un sistema haciendo uso de herramientas CASE.	Prácticas de la asignatura	Clientes Alumnos Responsable	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T5 Especificaciones formales de requisitos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas formales de especificación de sistemas, incluyendo la especificación de aspectos dinámicos y los métodos formales orientados a objetos	En el tema 6	Responsable de la asignatura	Transparencias Bibliografía básica	Ejercicios prácticos Examen

T7 Métodos de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Profundización en los métodos orientados a objetos. Estudio del Lenguaje Unificado de Modelado y del Proceso Unificado de Rational	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Ejercicios prácticos Defensa de la práctica Examen
Realización del análisis y diseño de un sistema orientado a objetos con la ayuda de una herramienta CASE	Práctica 2 de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudiar los procesos relacionados con la modificación de un producto software después de su entrega.	Tema 9	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

T13 Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las particularidades que presenta el desarrollo de sistemas de tiempo real y de las técnicas más apropiadas para el modelado de este tipo de sistemas	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Análisis de las características de los sistemas distribuidos y su relación con la aplicación de métodos de Ingeniería del Software.	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Examen de otros sistemas no convencionales como sistemas expertos, sistemas para el soporte a las decisiones, etc.	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Con la ayuda de herramientas CASE, los alumnos realizarán el análisis y diseño de sendos sistemas de software aplicando, respectivamente, un método estructurado y otro orientado a objetos.	Clases prácticas	Responsable de la asignatura Alumnos Clientes	Transparencias Bibliografía Herramientas CASE Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Promover los debates sobre las soluciones presentadas en los ejercicios realizados en clase	Clase de teoría	Responsable de la asignatura Grupos de alumnos	Pizarra Medios audiovisuales	Estudio del seguimiento de los ejercicios Opinión alumnos
Defensa oral de las prácticas	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Propuesta de ejercicios de aplicación de los conceptos teóricos	Durante las clases de teoría	Responsable de la asignatura Grupos de alumnos	Transparencias Pizarra	Evaluación de los ejercicios entregados
Documentación de las prácticas	Defensa de las prácticas	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de las prácticas en grupos de 2 personas	Durante las clases prácticas	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

A.3.5.5 Criterios de evaluación de la asignatura

a) Parte de Teoría

- Un examen final en el mes de junio.

- Un examen final en el mes de septiembre.

b) Parte Práctica

- En junio la evaluación de esta parte de la asignatura se realizará mediante la valoración de los trabajos presentados. La nota será la media de las obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individualizada.

- En septiembre se realizará un examen en el que el alumno tendrá que resolver un supuesto práctico con la ayuda de una herramienta CASE.

c) Trabajos voluntarios presentados (de *0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida*)

d) En el caso de haber superado la parte teórica y la práctica, se aplicará la siguiente fórmula para la obtención de la calificación final:

Nota Final = ((Teoría*0,6) + (Práctica*0,3))* 10/9 + Nota trabajos

A.3.5.6 Bibliografía básica de referencia

En el apartado de bibliografía se recogen aquellas referencias que pueden ayudar al alumno a completar los conocimientos impartidos en las clases. La elección de las mismas, como ocurre en otras asignaturas de la unidad, se ha basado en tres aspectos: *su adecuación a los contenidos, su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

🕮 **Booch, G., Rumbaugh, J. and Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

El texto en castellano corresponde a la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. Es uno de los libros de la trilogía publicada por los creadores del lenguaje modelado UML, en él se proporciona una referencia de uso de las características específicas de UML, centrándose fundamentalmente la notación gráfica. Es indispensable su consulta en el tema 4 y fundamentalmente en el 5.

🕮 **Graham, I.** "*Métodos Orientados a Objetos*". 2ª Ed. Addison-Wesley/Díaz de Santos, 1996.

Traducción del libro del mismo autor "*Object-Oriented Methods*" 2nd ed. Addison-Wesley, 1994. En un libro introductorio a la Orientación al Objeto. Cubre de forma amplia y correcta todos los aspectos de la Orientación al Objeto: conceptos básicos, lenguajes de programación, bases de datos, aplicaciones y métodos. En los capítulos de métodos de análisis y de diseño realiza una exposición y un examen de los de mayor difusión, cubriendo prácticamente todas las escuelas existentes. Incorpora un capítulo interesante relacionado con la inteligencia artificial y la teoría de los conjuntos difusos. Se considera un buen libro de referencia donde se puede adquirir una panorámica amplia de la Orientación al Objeto.

🕮 **Hatley, D. J. y Pirbhai, I. A.** "*Strategies for Real-Time System Specification"*. Dorset House Pub. Co., 1987.

Se describen los métodos para la especificación de los requisitos y el diseño de sistemas de tiempo real basados en computadora. Los métodos propuestos integran las herramientas del análisis estructurado, fundamentalmente el DFD, para la construcción del modelo de requisitos y del modelo arquitectónico. Partiendo de la base de los DFD propone una herramienta de modelado para los procesos de control los CFD (*Control Flow Diagrams*), y otra para la representación de la configuración física del sistema los ACF (*Architecture Flow Diagrams*). Los métodos y la construcción de modelos están apoyados por un ejemplo que sirve de hilo conductor a lo largo del libro. Aunque el método propuesto para la asignatura de Ingeniería del Software I va a seguir más la propuesta de Ward & Mellor, este libro tiene aportaciones muy interesantes en el modelado de sistemas de tiempo real, sobre todo en el modelado arquitectónico.

🕮 **Henderson-Sellers, B**. "*A Book of Object-Oriented Knowledge"*. 2nd ed. Prentice Hall PTR, 1997.

Es una introducción básica al enfoque orientado al objeto en la Ingeniería del Software. El libro describe las ideas básicas relacionadas con el análisis, el diseño y la implementación e incluye un buen conjunto de referencias para el estudio con profundidad de cada uno de esos temas. Aporta en cada tema el diseño de las transparencias que pueden ser utilizadas en la explicación del mismo. Es una buena presentación de los conceptos y los métodos de la Orientación al Objeto.

🕮 **Jacobson, I.; Booch, ; G. Rumbaught, J**. "*The Unified Software Development Process*". Addison-Wesley Object Technology Series, 1999.

Otro de los libros de la trilogía de los creadores de UML, en el que se ofrece una referencia completa del proceso de desarrollo unificado que se adapta a UML. Su consulta puede ser beneficiosa en el estudio de los temas 4 y 5.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Esta traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997 en su segunda edición supone una extensa revisión respecto a la primera. Se considera una referencia importante para el Tema 4.

🕮 **Piattini Velthuis, M. G., Calvo-Manzano, J. A., Cervera, J. y Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

Libro recomendado e la asignatura Ingeniería del Software de primer ciclo que puede ser útil también en el segundo para repasar y profundizar en algunos aspectos relacionados con: *Sistemas de Información* (capítulos 1 y 2), *El Proceso de Desarrollo de Software* (capítulos 3-16) y *Tecnología* (capítulos 17-19) que se tratan en las dos primeras unidades docentes de la asignatura.

📖 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill. 1998.

Este libro, aunque ya ha sido manejado ampliamente por los alumnos en primer ciclo, puede ser igualmente útil en las asignaturas de Ingeniería del Software de segundo ciclo ya que abarca la mayoría de los temas relacionados con esta materia, tratándolos con bastante profundidad. Cada tema se complementa además con un gran número de referencias de otras fuentes de información sobre el tema, incluyendo recursos disponibles en Internet. Corresponde a la traducción del libro del mismo autor "*Software Engineering: A Practitioner´s Approach*", 4th Edition, McGraw-Hill, 1997. Se considera un texto imprescindible de referencia para esta asignatura siendo una referencia interesante en la práctica totalidad de los temas.

📖 **Sommerville, Ian**. "*Software Engineering*". 5th Edition, Addison-Wesley. 1996.

En esta última edición de la obra de Sommerville, éste realiza una extensa revisión de la anterior edición incluyendo nuevos temas y modificando sustancialmente algunos de los existentes como los dedicados a la tecnología CASE y evolución del software. Los contenidos se adaptan a la recomendación curricular ACM/IEEE-CS91 para los módulos SE2 a SE5. Se considera una libro de consulta importante para la práctica totalidad de la asignatura.

📖 **Rational Software Corporation.** "*OMG Unified Modeling Language Specification. Version 1.3*". http://www.rational.com. Abril, 1999.

Esta documentación corresponde a la última versión del estándar UML en la que se hace una revisión de la versión 1.1, enfocada fundamentalmente a la resolución de inconsistencias y clarificación de ambigüedades. Esta documentación incluye un apartado dedicado a la semántica de UML en el que se presenta la semántica de los elementos de los modelos de objetos estáticos y dinámicos, además se describe de una manera semi-formal el metamodelo. En otra sección del documento denominada *guía de la notación UML* se ofrece la notación propuesta para la representación gráfica de los modelos, compuesta de nueve tipos de diagramas diferentes. La consulta de esta documentación es imprescindible en el tema 5.

📖 **Rumbaught, J.; Jacobson, I.; Booch, G.** "*The Unified Modeling Language. Reference Manual*". Addison-Weslwy Object Technology Series, 1999.

Manual completo de referencia de UML que completa la trilogía de libros sobre UML de los tres autores. En el libro se expone en primer lugar la forma de llevar a cabo el modelado y los diagramas utilizados en cada una de las vistas del sistema. Posteriormente aparece en orden alfabético la descripción de todos los componentes del lenguaje. El libro puede ser de gran ayuda en el estudio de los temas 4 y 5, especialmente en éste último.

📖 **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

La versión en castellano de este libro corresponde a la traducción del texto del mismo autor "*Modern Structured Analysis*", Prentice-Hall, 1989. En él se describe el método propuesto por Yourdon que se basa en las ideas de DeMarco en el año 1979. Recomendado como referencia importante en la asignatura de Ingeniería del Software de primer ciclo, puede ser de ayuda en esta asignatura para el estudio del tema 3.

A.3.6 Administración de Proyectos Informáticos

El propósito fundamental de esta asignatura es proporcionar los conocimientos y capacidades prácticas necesarias para llevar a cabo las actividades de Ingeniería del Software relacionadas con los aspectos de gestión y control de proyectos. Dichas tareas comienzan antes del inicio de cualquier actividad técnica y continúan a lo largo de todo el ciclo de vida del software. El objetivo final de dichas actividades es obtener un producto software final de calidad y fiable desarrollado mediante un proceso eficiente y productivo.

Los objetivos de la unidad docente que se pretenden conseguir en esta asignatura son los siguientes:

T10, T11, T12, P3, P8, H1, H2, H3 y **H4**.

Los apartados que se exponen a continuación siguen la pauta marcada en las asignaturas precedentes.

A.3.6.1 Ficha de la asignatura

Asignatura	*Administración de Proyectos Informáticos (troncal)*
Créditos	*6T + 3P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*2º*
Cuatrimestre	*1º y 2º (anual)*
Responsable	*María N. Moreno García (mmg@gugu.usal.es)*
Página web de la asignatura	*http://olivo.usal.es/~mmoreno/api.html*

Tabla A.18. Administración de Proyectos Informáticos

A.3.6.2 Prerrequisitos

Para lograr los objetivos anteriores es fundamental que el alumno haya adquirido previamente conocimientos sobre el proceso de desarrollo, ya que dichas nociones son básicas para la comprensión y asimilación de los conceptos y métodos que se tratan en esta asignatura. Los conocimientos requeridos se adquieren en las asignaturas **Ingeniería del Software** que se imparte en 3º de ITIS y **Análisis de Sistemas** perteneciente a primer curso del segundo ciclo.

A.3.6.3 Temario teórico/práctico

Presentación de la asignatura (1 hora)
Tema 1. Visión general de la administración de proyectos (5 horas)
Tema 2. Medición del software (12 horas)
Tema 3. Métodos de estimación y gestión del riesgo (10 horas)
Tema 4. Planificación temporal de proyectos (12 horas)
Tema 5. Gestión de la calidad (15 horas)
Tema 6. Gestión de configuraciones (5 horas)

Tabla A.19. Programa teórico de la asignatura Administración de Proyectos Informáticos (6 créditos)

Laboratorio: uso de herramientas automatizadas (30 horas)

Práctica 1. Aplicación de métricas (4 horas)
Práctica 2. Estimación de coste y esfuerzo de un proyecto (8 horas)
Práctica 3. Planificación temporal del proyecto (18 horas)

Tabla A.20. Programa práctico de la asignatura Administración de Proyectos Informáticos (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas

En la Tabla A.21 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T10, T11, T12, H3
Tema 2	T11, T4, H3
Tema 3	T10, H3
Tema 4	T10, H3
Tema 5	T10, T11, H3
Tema 6	T12, H3
Práctica 1	P8, H1, H2, H4
Práctica 1	P3, H1, H2, H4
Práctica 2	P3, H1, H2, H4

Tabla A.21. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.6.4 Líneas de acción

Como en las asignaturas anteriores, seguidamente se comentarán todos los componentes de las líneas de acción que contribuirán a la consecución de los objetivos.

T10 Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

T11 Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Ofrecer una visión general de las actividades de gestión, planificación y seguimiento de proyectos	En el tema 1	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Mostrar el alcance de las métricas del software y las dos vertientes de su aplicación: evaluación y predicción	En el tema 2 y en las prácticas de la asignatura	Responsable de la asignatura Alumnos	Transparencias Bibliografía básica	Defensa de la práctica Examen
Estudiar y aplicar los conceptos y métodos relacionados con las actividades descritas en el primer tema.	Todos los temas de la asignatura Prácticas de la asignatura	Responsable de la asignatura Alumnos	Transparencias Bibliografía básica	Defensa de las prácticas Examen

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Examinar los conceptos y tareas asociados a la gestión de configuraciones, acentuando la importancia de asegurar la calidad y la consistencia de los cambios.	Tema 6	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P3 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Utilización de herramientas automatizadas de gestión para la planificación y la estimación de costes.	En las clases prácticas	Responsable de la asignatura Alumnos	Medios audiovisuales Herramientas de gestión. Documentación de trabajos realizados el curso anterior	Defensa de las prácticas

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Utilización de herramientas automatizadas para la aplicación de métricas.	En las clases prácticas	Responsable de la asignatura Alumnos	Herramientas de gestión. Documentación de trabajos realizados el curso anterior	Defensa de las prácticas

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Fomentar la participación en las clases de problemas	A lo largo del curso	Responsable de la asignatura Alumnos	Pizarra	Participación de los alumnos
Defensa oral de las práctica	Defensa de la práctica	Alumnos	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Propuesta de ejercicios de aplicación de los conceptos teóricos	Durante las clases de teoría	Responsable de la asignatura alumnos	Transparencias Pizarra	Evaluación de los ejercicios entregados
Documentación de las prácticas	Defensa de las prácticas	Alumnos Responsable de la asignatura	Herramientas de gestión Bibliografía	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de las prácticas en grupos de 2 personas	Durante las clases prácticas	Alumnos Responsable de la asignatura	Documentación de trabajos realizados el curso anterior	Defensa de la práctica

A.3.6.5 Criterios de evaluación de la asignatura

a) Parte de Teoría

- Un examen final en el mes de junio.

- Un examen final en el mes de septiembre.

b) Parte Práctica

- En junio la evaluación de esta parte de la asignatura se realizará mediante la valoración de los trabajos presentados. La nota será la media de las obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individualizada.

- En septiembre se realizará un examen en el que el alumno tendrá que resolver un supuesto práctico con la ayuda de una herramienta.

c) Trabajos voluntarios presentados (de *0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida*)

d) En el caso de haber superado la parte teórica y la práctica, se aplicará la siguiente fórmula para la obtención de la calificación final:

Nota Final = ((Teoría*0,6) + (Práctica*0,3))* 10/9 + Nota trabajos

A.3.6.6 Bibliografía básica de referencia

Siguiendo la misma línea que en el resto de las asignaturas de la unidad docente, la bibliografía que aquí se incluye servirá de complemento al material suministrado para el seguimiento de las clases. Los criterios seguidos para su elección son los mismos que en los casos anteriores.

🕮 **Burnett, K.** "*The Project Management Paradigm*". Springer, 1998.

Este texto proporciona un enfoque de la gestión de proyectos basado en la combinación de aspectos humanos y principios metodológicos para conseguir el éxito del proyecto. Es interesante la visión que ofrece sobre los conceptos, principios y métodos de estimación y gestión de riesgos. El capítulo que dedica a la gestión de la calidad es bastante completo, por lo que puede ser muy útil en el estudio del tema 5.

🕮 **Cos, M.** "*Teoría General del Proyecto*". Síntesis, 1997.

El primer volumen de este texto cubre los conceptos y métodos de organización de proyectos, estimación, planificación y control. El último capítulo está dedicado al plan de calidad del proyecto

🕮 **Fenton, N.E. y Pfleeger, S.L.** "*Software Metrics. A Rigorous & Practical Approach*". PWS 1997.

Excelente obra sobre métricas del software que cubre diversos aspectos de su aplicación como son la evaluación y predicción del tamaño y funcionalidad del software, de la productividad, de costes y esfuerzos y de la calidad del producto y del proceso de desarrollo. En el se recogen gran número de métricas existentes en bibliografía así como algunas propuestas por los mismos autores del libro. Los primeros capítulos dedicados a los fundamentos de la medición y a la forma de realizar medidas eficaces son muy recomendables. Imprescindible en el estudio de los temas 2, 3 y 5.

🕮 **McConnell, S.** "*Desarrollo y Gestión de Proyectos Informáticos*". McGraw Hill, 1997.

Traducción de la primera edición en inglés de "*Rapid Development*". Este libro, dirigido principalmente a las estrategias de desarrollo rápido, dedica el capítulo 4 a explicar de manera clara las bases del desarrollo de software. Tiene también temas dedicados a la estimación y gestión de riesgos. Los capítulos 11, 12 y 13 constituyen una buena referencia sobre los aspectos a considerar en la formación de equipos de trabajo (motivación, características, estructura...), al igual que el capítulo 15 en el apartado de herramientas de mejora de la productividad. La tercera parte del libro, dedicada a "métodos recomendables" cubre la práctica totalidad de los temas de la asignatura.

🕮 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGrawHill. 1998.

Este clásico de la Ingeniería del Software constituye una referencia obligada en casi todos los temas de la asignatura. El capítulo 3 trata de forma global los aspectos de gestión del proyecto. La medición de software aparece en apartados sueltos incluidos en diferentes capítulos y en un tema relacionado con el proceso software que está dedicado casi completamente a la medición. El resto de los temas de la asignatura los contempla también de forma detallada en diferentes capítulos.

🕮 **Puig, J.** "*Proyectos Informáticos. Planificación, Desarrollo y Control*", Paraninfo, 1994.

El texto de Puig puede ser de utilidad para obtener una visión global de las actividades clásicas de gestión de proyectos que se tratan de una manera concisa y clara. El libro utiliza, en algunos casos, notaciones y terminología obsoleta y no incluye técnicas actuales, por lo que necesitaría una revisión.

🕮 **Quang, P. Y Gonin J.** "*Dirección de Proyectos Informáticos*". Eyrolles, 1994.

En este libro se contemplan los métodos clásicos de organización de equipos, estimación, planificación, seguimiento y control de calidad de un proyecto. Además se ofrecen una serie de recomendaciones obtenidas de experiencias prácticas. El libro nos proporciona una visión superficial de los temas de esta asignatura, por lo que se aconseja su uso únicamente en el tema de introducción.

🕮 **Romero, C.** "*Técnicas de Programación y Control de Proyectos*". Pirámide, 1997

Describe de forma detallada tres métodos clásicos de planificación temporal (PERT, CPM y ROY) partiendo de la explicación previa de los conceptos sobre teoría de grafos que son la base de tales métodos. Puede ser de gran ayuda en el estudio del tema 4.

🕮 **Sommerville, I.** "*Software Engineering*". 5th Edition. Addison-Wesley, 1996.

La quinta edición de esta obra sobre ingeniería del software incorpora nuevos capítulos sobre temas de gestión (personal, calidad, etc.) y mejora los dedicados a la tecnología CASE y evolución del software. Estos cambios lo convierten un libro de consulta indispensable de la asignatura.

A.3.7 Sistemas de Información

El propósito fundamental de esta asignatura es la aplicación práctica de los principios de la Programación Orientada a Objetos estudiados en otras asignaturas.

Los objetivos de la unidad docente que se pretenden conseguir en esta asignatura son los siguientes:

P7, H1, H2, H3 y **H4**.

Los apartados que se exponen a continuación siguen la pauta marcada en las asignaturas precedentes.

A.3.7.1 Ficha de la asignatura

Asignatura	*Sistemas de Información (troncal)*
Créditos	*0T + 9P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*2º*
Cuatrimestre	*1º*
Responsable	*José Rafael García-Bermejo Giner (coti@gugu.usal.es)*
Página web de la asignatura	*http://acebuche.usal.es*

Tabla A.22. Sistemas de Información

A.3.7.2 Prerrequisitos

Para lograr los objetivos anteriores es fundamental que el alumno haya adquirido previamente conocimientos sobre la realización de proyectos software, tanto en la asignatura de **Ingeniería del Software** del tercer curso del primer ciclo, como en la asignatura de **Análisis de Sistemas** del primer curso del segundo ciclo.

Además, sería aconsejable que el alumno tuviera conocimientos prácticos de **Programación Orientada a Objetos**, del tercer curso del primer ciclo.

A.3.7.3 Temario práctico

Presentación de la asignatura (1 hora)
Tema 1. Conceptos básicos de orientación a objetos (9 Horas)
Tema 2. El lenguaje Java. Java frente a C/C++ (20 Horas)
Tema 3. Applets (15 Horas)
Tema 4. Creación de aplicaciones en Java (15 Horas)
Tema 5. Realización de un supuesto práctico (30 Horas)

Tabla A.23. Programa de prácticas (9 Créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos.
- Trabajos voluntarios realizados por los alumnos.
- Conferencias invitadas.

A.3.7.4 Líneas de acción

P7 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Búsqueda e instalación del entorno de desarrollo más adecuado y reciente para su uso en la asignatura	En la fase inicial del curso	Responsable de la asignatura Alumnos	Internet Revistas Bibliografía	
Cambio de mentalidad para la adecuada aplicación del paradigma objetual	A partir del tema 2	Responsable de la asignatura	Numerosos ejercicios prácticos	Práctica en dos ocasiones a lo largo del curso y examen teórico

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa en seminarios de algún tema teórico o ejercicio práctico	A lo largo del curso	Responsable de la asignatura alumnos	Audiovisuales	Participación de los alumnos

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Extracción y documentación de las especificaciones	En el desarrollo de los trabajos prácticos	Alumnos	Enunciados de los problemas	Evaluación de los ejercicios entregados

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Búsqueda de información actualizada en Internet	A lo largo de la asignatura	Alumnos	Internet Revistas	Resultados de los ejercicios prácticos

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Construcción de aplicaciones modulares, realizando los distintos grupos los módulos componentes	Durante las clases prácticas	Alumnos Responsable de la asignatura	Enunciado Entorno de desarrollo	Por la interfaz y la implementación de los módulos

A.3.7.5 Criterios de evaluación de la asignatura

a) Parte Práctica

- En diciembre y en febrero se examinan los ejercicios realizados.
- En febrero se realiza un examen escrito.
- En septiembre se realizará un examen escrito.

c) Se aplica la siguiente fórmula para la obtención de la calificación final:

Nota Final = (Ejercicios entregados*0,5) + (Examen*0,5)

A.3.7.6 Bibliografía básica de referencia

Siguiendo la misma línea que en el resto de las asignaturas de la unidad docente, la bibliografía que aquí se incluye servirá de complemento al material suministrado para el seguimiento de las clases.

- **Jaworski, Jaime**. "*Java 1.2 Unleashed*". Sams, 1998.
- **Cadenhead, Rogers**. "*Teach Yourself JAVA 1.2 in 24 hours*". Sams, 1999.
- **Lemay, Laura Lemay and Cadenhead, Rogers**. "*Teach Yourself Java 2 Platform in 21 Days: Professional Reference Edition*". Sams Teach Yourself in 21 Days Series. Sams, 1999.
- **SUN Microsystems.** "*Java™ Standard Edition Platform Documentation*". http://java.sun.com/docs/index.html. [Última vez visitado, 16-3-2000]. 2000.
- **SUN Microsystems.** "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16-3-2000]. February, 2000.

A.3.8 Proyecto I.I

El proyecto de la Ingeniería Informática se convierte de nuevo en otra excelente ocasión para la consecución del objetivo **P9**, pero esta vez con la experiencia de haber realizado previamente otro proyecto y haber cursado dos años más estudios universitarios de segundo ciclo.

Los objetivos de este proyecto no deben ser un calco de los del proyecto de I.T.I.S, sino que debe convertirse en un trabajo donde el alumno demuestre su madurez, además de sus conocimientos y dominio de la técnica. Así, el proyecto debe ir acompañado de las métricas oportunas, planes de implantación y mantenimiento, justificaciones teóricas basadas en el estado del arte del tema abordado...

En este sentido la guía recogida en [García et al., 1999a] sigue siendo una referencia de consulta válida, pero ha de tenerse en cuenta que dicha guía fue desarrollada para los proyectos de I.T.I.S, no para un proyecto de un segundo ciclo.

A.4 Referencias

[Buxton et al., 1976] Buxton, J. M., Naur, P. and Randell, B. (eds.) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[García et al., 2000] García Peñalvo, Francisco José, Maudes Raedo, Jesús Manuel, Piattini Velthuis, Mario Gerardo, García-Bermejo Giner, José Rafael y Moreno García, María N. "*Proyecto de Final de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación*". Departamento de Informática y Automática de la Universidad de Salamanca. Versión 1.5.2. http://tejo.usal.es/~fgarcia/doc/pfc.pdf. Marzo, 2000.

[García et al., 1999b] García Peñalvo, Francisco José, Moreno García, María N., González Talaván, Guillermo y Moreno Montero, Ángeles María. "*Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 46 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 1999c] García Peñalvo, Francisco José, Moreno García, María N., Montero García, Eduardo y Arranz Val, Pablo. "*Evaluación del Profesorado: Un Protocolo de Evaluación por Pares*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 47 y ponencia en versión digital (CD-ROM). 1999.

[Leveson, 1997] Leveson, Nancy G. "*Software Engineering: Stretching the Limits of Complexity*". Communications of the ACM 40(2): 129-131. February, 1997.

[Parnas, 1997] Parnas, David L. "*Software Engineering: An Unconsummated Marriage*". Communications of the ACM 40(9): 128. September, 1997.

[Shaw and Tomayko, 1991] Shaw, Mary and Tomayko, James E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10). Software Engineering Institute – Carnegie Mellon University. Pittsburgh, Pennsylvania 15213 (USA). August, 1991.

Apéndice B
Acrónimos

1FN	**Primera Forma Normal.**
2FN	**Segunda Forma Normal.**
3FN	**Tercera Forma Normal.**
4FN	**Cuarta Forma Normal.**
4GL	**Fourth Generation Language.**
5FN	**Quinta Forma Normal.**
ABF	**Applications By Forms.**
ACF	**Architecture Flow Diagram.**
ACF	**Asset Certification Framework.**
ACM	**Association for Computing Machinery.**
ACTI	**Abstract and Concrete Templates and Instances.**
ADAGE	**Avionics Domain Application Generation Environment.**
ADL	**Architecture Description Language.**
ADOO	**Análisis y Diseño Orientado al Objeto.**
ADP	**Acyclic Dependencies Principle.**
ADP	**Automated Data Processing.**
ADS	**Application Development Strategies.**
ADT	**Abstract Data Type.**
ADT	**Application Development Trends.**
AECC	**Asociación Española para el Control de Calidad.**
AEIA	**Asociación Española de Informática y Automática.**
AENOR	**Asociación Española de Normalización y Certificación.**
AF	**Applications Frames.**
AF	**Auditoría Funcional.**
AFFTC	**Air Force Flight Test Centre.**
AFI	**Auditoría Física.**
AHP	**Analytic Hierarchy Process.**
AI	**Artificial Intelligence.**
AII	**Asociación de Ingenieros en Informática.**
AIP	**Atributo Identificador Principal.**
AIS	**Automated Information Systems.**
AKO	**A Kind Of.**
ALF	**Asset Library Framework.**
ALGOL	**ALGOrithmic Language.**
ALI	**Asociación de Doctores, Licenciados e Ingenieros en Informática.**
ALOAF	**Asset Library Open Architecture Framework.**
ALS	**ADA Language System.**
ANSI	**American National Standard Institute.**
AO	**Application Objects.**
AOO	**Análisis Orientado al Objeto.**
APD	**Agencia de Protección de Datos.**
API	**Application Programming Interface.**
APO	**A Part Of.**
ARC	**Army Reuse Center.**
ARM	**Annotated Reference Manual.**
ARPANet	**Advanced Research Project Agency.**
ARS	**Análisis Requisitos del Sistema.**
ARSi	**Actividad i del Módulo de Análisis de Requisitos del Sistema.**
ASC	**Aeronautical Systems Center.**
ASCII	**American Standard Code for Information Interchange.**
ASIS	**Ada Semantic Interface Specification.**
ASSET	**Asset Source for Software Engineering Technology.**
ATI	**Asociación de Técnicos en Informática.**
ATIS	**A Tool Interface Standard.**
ATL	**Active Template Library.**
ATM	**Asynchronous Transfer Mode.**
AVEIN	**Asociación Vallisoletana de Empresas de Informática.**
AWT	**Abstract Windows Toolkit.**
BAe	**British Aerospace.**

BBS **Bulletin Board System.**
BCNF **Boyce/Codd Normal Form.**
BD **Bases de Datos.**
BDE **Borland Database Engine.**
BDK **Beans Development Kit.**
BDOO **Bases de Datos Orientadas a Objetos.**
BIDM **Basic Interoperability Data Model.**
BIT **Binary digIT.**
BITNET **Because IT's time NETwork.**
BLOB **Bynary Large Object.**
BLOOM **BarceLona Object Oriented Model.**
BNF **Backus Naur Form.**
BOA **Business Object Arquitecture.**
BOE **Boletín Oficial del Estado.**
BOF **Business Object Facility.**
BOM **Business Object Model.**
BON **Business Object Notation.**
BOO **Behavioural Object-Oriented.**
BPR **Business Process Reengineering.**
BQL **Binary Query Language.**
BRM **Binary Relations Model.**
BSA **Business Software Alliance.**
BSI **British Standards Institute.**
BTW **By The Way.**
C/S **Cliente/Servidor.**
CA **Certificartion Authority.**
CACM **Communications of the ACM.**
CACS **Computer Audit, Control and Security Conferences.**
CAD **Computer Aided Design.**
CADF **Comimittee for Advanced Database Function.**
CADMAT **Computer Aided Design Manufacturing and Testing.**
CADSAT **Computer Aided Design and Specification Analysis Tool.**
CAI **Computer Assisted Instruction.**
CAL **Computer Aided Learning.**
CAM **Computer Aided Manufacturing.**
CAMAC **Computer Automated Measurement and Control.**
CAMP **The Common Ada Missile Packages project.**
CARDS **Comprehensive Approach to Reusable Defense Software**
CARE **Computer-Aided Requirements Engineering.**
CAS **Commercially Available Software.**
CASE **Computer Aided/Assisted Software/System Engineering.**
CASRE **Computer Aided Software Reliability Estimation.**
CBD **Component Based Development.**
CBL **Computer Based Learning.**
CBO **Common Business Objects.**
CBO **Coupling Between Object classes.**
CBS **Compound Boolean Selection.**
CBSE **Component-Based Software Engineering.**
Cc **Código Civil.**
CCA **Critical Capability Area.**
CCB **Change Control Board.**
CCC **Comité de Control de Cambios.**
CCCP **Cliente-Canal-Compañía-Producto.**
CCITT **Consultative Committee International Telephone and Telegraph.**
CCP **Common Closure Principle.**
CCTA **Central Computing and Telecommunications Agency.**
CD **Compact Disc.**
CDE **Common Desktop Environment.**
CDIF **CASE Data Interchange Format.**
CDM **Common Data Model.**
CDR **Critical Design Review.**
CDRL **Contract Data Requirements List.**
CE **Comunidad Europea.**
CEC **Configuración de Estado de la Configuración.**
CEF **Centro de Estudios Financieros.**
CEN **Comisión Europea de Normalización.**
CEPIS **Council of European Professional Informatics Societies.**
CETE **Centre d'Etudes Techniques de l'Equipement,**
CF **Common Facilities.**
CFD **Control Flow Diagram.**
CFE **Control Flow Editor.**
CFRP **Conceptual Framework for Reuse Processes.**
CGI **Common Gateway Interface.**
CICS **Customer Information Control System.**
CIM **Computer Integrated Manufacturing.**
CISA **Certified Information Systems Auditor.**
CLLCM **Clear Lake Lifecycle Model.**
CLOS **Common Lisp Object System.**
CLP **Construcción Lógica de Programas.**
CM **Configuration Management.**
CMM **Capability Maturity Model.**
COBOL **COmmon Bussines Oriented Language.**
COCOMO **Cost Constructive Model.**
CODASYL **Conference On DAta SYstems Languages.**
COM **Component Object Model.**
COMMA **Common Object-oriented Methodology Metamodel Architecture.**
COMN **Common Object Modelling Notation.**
COP **Component-Oriented Programming.**
CORBA **Common Object Request Broker Architecture.**
COSE **Common Open Software Environment.**
COTS **Commercial Off-The-Shelf.**
CP **Código Penal.**

CPD	**Centro de Proceso de Datos.**
CPI	**Centro Proveedor de Información.**
CPM	**Critical Path Method.**
CPU	**Central Processing Unit.**
CR	**Configuración de Referencia.**
CRC	**Cyclic Redundancy Checking.**
CRC	**Class, Responsability and Collaboration.**
CS&S	**Computer Systems and Software.**
CSC	**Computer Software Component.**
CSCI	**Computer Software Configuration Item.**
CSF	**Critical Success Factors.**
CSPEC	**Control SPECification.**
CSU	**Computer Software Unit.**
CTI	**Centre Technique d'Informatique.**
CTN	**Comité Técnico Nacional.**
CUT	**Classes Under Test.**
CWA	**Closed World Assumption.**
DA	**Data Administrator.**
DA	**Domain Analysis.**
DACS	**Data & Analysis Center for Software.**
DAFTG	**Database Arquitecture Framework Task Group.**
DAG	**Directed Acyclic Graph.**
DALE	**Database Access Library for Eiffel.**
DARPA	**Defense Advanced Research Project Agency.**
DARTS	**Design Approach for Real-Time Systems.**
DAS	**Documento de Análisis del Sistema.**
DBA	**Database Administrator.**
DBCC	**Database Consistency Checker.**
DBMS	**Data Base Management System.**
DBSSG	**Database System Study Group.**
DBTG	**Data Base Task Group.**
DCC	**Diagrama de Configuración de Clases.**
DCE	**Diagrama de Correspondencia del Efecto.**
DCE	**Distributed Computing Environment.**
DCL	**Data Control Language.**
DCMC	**Defense Contract Management Command.**
DCOM	**Distributed Component Object Model.**
DCS	**Módulo de Desarrollo de Componentes del Sistema.**
DCSi	**Actividad i del Módulo de Desarrollo de Componentes del Sistema.**
DD	**Data Dictionary.**
DDBMS	**Distributed Data Base Management System.**
DDC	**Display Data Channel.**
DDF	**Diagramas de Descomposición Funcional.**
DDL	**Data Definition Language.**
DDM	**Data Dictionary Manager.**
DDS	**Data Dictionary System.**
DDS	**Digital Data Storage.**
DE/R	**Diagrama de Entidad-Interrelación.**
DEC	**Diagrama de Estados de Clases.**
DED	**Diagrama de Estructura de Datos.**
DEL	**Diagrama de Línea de Ensamblaje.**
DER	**Diagrama de Entidad-Interrelación.**
DES	**Data Encryption Standard.**
DF	**Dependencia Funcional.**
DFD	**Diagrama de Flujo de Datos.**
DFE	**Data Flow Editor.**
DFOQ	**Design For Ownership Quality.**
DFS	**Distributed File System.**
DIB	**Device Independent Bitmap.**
DIO	**Diagrama de Interacción entre Objetos.**
DIP	**Dependency Inversion Principle.**
DIS	**División de Ingeniería del Software.**
DIT	**Depth of Inheritance Tree.**
DL	**Data Language.**
DLL	**Dynamic-Link Libraries.**
DMA	**Direct Memory Access.**
DMCS	**Data Mapping Control System.**
DML	**Data Manipulation Language.**
DMT	**Data Management Tool.**
DNS	**Domain Name System.**
DoD	**Departament of Defense.**
DOE	**Distributed Object Everywhere.**
DOME	**Distributed Object Management Everywhere.**
DOO	**Diseño Orientado al Objeto.**
DOR	**Domain Oriented Reuse.**
DPU	**Módulo de Desarrollo de Procedimiento de Usuario.**
DPUi	**Actividad i del Módulo de Desarrollo de Procedimiento de Usuario.**
DR	**Deficiency Report.**
DRA	**Desarrollo Rápido de Aplicaciones.**
DRI	**Diccionario de Recursos de Información.**
DRS	**Documento de Requisitos del Sistema.**
DSDL	**Data Storage Definition Language.**
DSE	**Data-Structure Editor.**
DSED	**Desarrollo de Sistemas Estructurados en Datos.**
DSI	**Director de Sistemas de Información.**
DSIC	**Departamento de Sistemas Informáticos y Computación.**
DSMC	**Defense Systems Management College.**
DSN	**Data Source Name.**
DSOM	**IBM's Distributed Systems Object Model.**
DSS	**Decision Support System.**
DSSA	**Domain-Specific Software Architecture.**
DTC	**Distributed Transaction Coordinator.**
DTE	**Diagrama de Transición de Estados.**
DTIC	**Defense Technical Information Center.**
DTS	**Módulo de Diseño Técnico del Sistema.**
DTSi	**Actividad i del Módulo de Diseño Técnico del Sistema.**
DVD	**Digital Versatile Disk.**

DW **Data Warehouse.**
E/R **Entity-Relationship.**
E/S **Entrada/Salida.**
EAP **Experimental Aircraft Programme.**
ECA **Event-Condition-Action rules.**
ECMA **European Computer Manufacterer's Association.**
ECS **Embedded Computer Systems.**
EDCS **Evolutionary Design of Complex Systems.**
EDI **Electronic Data Interchange.**
EDIF **Electronic Data Interchange Format.**
EDS **Entornos de Desarrollo de Software.**
EET **Earliest Even Time.**
EFF **Electronic Frontier Foundation.**
EFS **Especificación Funcional del Sistema.**
EFSi **Actividad i del Módulo de Especificación Funcional del Sistema.**
EIA **Electronic Industries Association.**
EIS **Executive Information System.**
EITO **European Information Technology Observatory.**
ELH **Entity Life History.**
ELP **Estructura Lógica del Proceso.**
ELS **Estructura Lógica de la Salida.**
ELSA **Electronic Library Services and Applicactions.**
EMS **Enterprise Messaging Server.**
ENCORE **Extensible and Natural Common Object REsource.**
ERD **Entity-Relationship Diagram.**
ERE **Entity-Relationshio Editor.**
EROOS **Entity-Relationship Object-Oriented Specificattions.**
ERS **Especificación de Requisitos del Software.**
ESA **European Space Agency.**
ESI **European Software Institute.**
ESPRIT **European Strategic Programme for Research and development in Information Technology.**
ESSI **European Systems & Software Initiative.**
ETS **Educational Testing Service.**
EUROWARE **Enabling Users to Reuse Over Wide AREas.**
F^3 **From Fussy to Formal.**
FA **Factor de Ajuste.**
FAQ **Frequently Asked Questions.**
FBE **Framework-Based Environment.**
FC **Factor de Complejidad.**
FCM **Factor Criteria Metrics Model.**
FCT **Factor de Complejidad Técnica.**
FD **Flujo de Datos.**
FESABID **Federación Española de Sociedades de Archivística, Biblioteconomía y Documentación.**
FESI **Federación Española de Sociedades de Informática.**
FH **Funcionalidad Hipermedia.**
FI **Facultad de Informática.**
FIFO **First In First Out.**
FMOODS **Formal Methods for Open Object-based Distributed Systems.**
FN **Forma Normal.**
FNBC **Forma Normal de Boyce/Codd.**
FODA **Feature-Oriented Domain Analysis.**
FOOD **Functional Object-Oriented Design.**
FOOL **Foundations of Object-Oriented Languages.**
FORTRAN **FORmula TRANslator.**
FP **Fundamental Principles.**
FPA **Function Point Analysis.**
FS **Feasibility Study.**
FSM **Finite State Machines.**
FTP **File Transfer Protocol.**
FYI **For Your Information.**
F-ORM **Funcionality in the Objects with Roles Model.**
GAD **Grafo Acíclico Dirigido.**
GAF **Generic Application Frame.**
GCS **Gestión de la Configuración del Software.**
GDI **Graphics Device Interface.**
GEARS **Gaining Efficiency and quAlity in Real time control Software.**
GI **Grado de Influencia.**
GIF **Graphics Interchange Format.**
GIRO **Grupo de Investigación en Reutilización y Orientación a Objetos.**
GNAT **GNU and New York University Ada Translator.**
GoF **Gang of Four.**
GOOD **General Object-Oriented Design.**
GOTS **Government Off-The-Shelf.**
GQM **Goal-Question-Metric.**
GRAPE **Graphical Programming for Eiffel.**
GSM **Global System for Mobile Communications.**
GTE **Gestión de la Transición de Estado.**
GUI **Graphics User Interface.**
HAL **Hardware Abstraction Layer.**
HDTV **High Definition TeleVision.**
HID **Human Interface Device.**
HIPO **Hierarchical Input Process Output.**
HLL **High Level Languages.**
HOOD **Hierarchical Object-Oriented Design.**
HP **Hewlett-Packard.**
HPCC **High Performance Computation & Communication**
HTML **HyperText Markup Language.**
HTTP **HyperText Transfer Protocol.**
HVE **Historia de la Vida de la Entidad.**

I/O	**Input/Output.**
IA	**Inteligencia Artificial.**
IAA	**Insurance Application Architecture.**
IAB	**Internet Architecture Board.**
IAC	**Information Analysis Center.**
IANA	**Internet Assigned Numbers Administration.**
IC	**Ingeniería del Conocimiento.**
ICASE	**Integrated CASE.**
ICI	**Instituto de Cooperación Iberoamericana.**
ICSR	**International Conference on Software Reuse.**
ICT	**Institut Catalá de Tecnología.**
IDD	**Interface Design Document.**
IDE	**Integrated Development Environment.**
IDeA	**Intelligent Design Aid.**
IDEAL	**Interactive Development Environment for Active Learning.**
IDL	**Interface Definition Language.**
i-DL	**Internal Data Language.**
IDO	**Identificador De Objeto.**
IE	**Information Engineering.**
IE	**Internet Explorer.**
IEC	**International Electrotechnical Commission.**
IEE	**Informáticos Europeos Expertos.**
IEEE	**Institute of Electrical and Electronics Engineers**
IFIP	**International Federation for Information Processing.**
IFPUG	**International Function Point User Group.**
IIS	**Microsoft Internet Information Server.**
IJG	**Independent JPEG Group.**
IMA	**Interactive Multimedia Association.**
IMHO	**In My Humble Opinion.**
IMLS	**Intelligent Multimedia Learning System.**
IMS	**Information Management System.**
IOOM	**Ithaca Object-Oriented Metodology.**
IP	**Internet Protocol.**
IPSE	**Integrated Programming Support Environment.**
IRC	**Internet Relay Chat.**
IRD	**Information Resource Dictionary.**
IRDS	**Information Resource Dictionary System.**
IRP	**Information Resource Planning.**
IRS	**Interface Requirements Specification.**
IS	**Information System.**
IS	**Ingeniería del Software.**
ISAC	**Ingeniería del Software Asistida por Ordenador.**
ISACA	**Information System Audit and Control Association.**
ISC	**Information Systems Contract.**
ISDN	**Integrated Services Data Network.**
ISE	**Interactive Software Engineering.**
ISM	**Integrated System Management.**
ISO	**International Standards Organization.**
ISP	**Interface Segregation Principle.**
ISS	**Integral Software Systems.**
ISV	**Independent Software Vendors.**
ITHACA	**Integrated Toolkit for Highly Advanced Computers Applications.**
ITIG	**Ingeniería Técnica en Informática de Gestión.**
ITIS	**Ingeniería Técnica en Informática de Sistemas.**
ITMRA	**Information Technology Management Reform Act.**
JAD	**Joint Application Design.**
JAL	**Java Algorithm Library.**
JCL	**Job Control Language.**
JDBC	**Java DataBase Connectivity.**
JDK	**Java Development Kit.**
JECF	**Java Electronic Commerce Framework.**
JGSE	**Joint Group on System Engineering.**
JIT	**Just In Time.**
JOOP	**Journal of Object-Oriented Programming.**
JPEG	**Joint Photographic Experts Group.**
JRP	**Joint Requirement Planning.**
JSD	**Jackson Structured Design.**
JSD	**Jackson System Development.**
JSP	**Jackson Structured Programming.**
JTC	**Joint Technical Commitee.**
JVM	**Java Virtual Machine.**
KBAS	**Knowledge Based Software Assistant.**
KLCD	**Miles de Líneas de Código.**
KPA	**Key Process Areas.**
KWIC	**KeyWord In Context.**
KWOC	**KeyWord Out Context.**
L4G	**Lenguaje de Cuarta Generación.**
LAN	**Local Area Network.**
LaSSIE	**Large Software System Information Environment.**
LC	**Lógica de Control.**
LCD	**Líneas de Código.**
LCM	**Language for Conceptual Modeling.**
LCOM	**Lack of Cohesion of Methods.**
LCP	**Leyes de Construcción de Programas.**
LDD	**Lenguaje de Definición de Datos.**
LDP	**Lenguaje de Diseño de Programas.**
LEP	**Lenguaje de Especificación Procedimental.**
LET	**Latest Even Time.**
LFN	**Long File Name.**
LIFO	**Last In First Out.**
LIL	**Library Interconnection Language.**
LLNL	**Lawrence Livermore National Laboratory.**
LOB	**Line Of Business.**
LOC	**Lines Of Code.**
LOO	**Lenguajes Orientados a Objetos.**

LORTAD **Ley Orgánica de Regulación del Tratamiento Automatizado de los Datos de carácter personal.**
LPI **Ley de Protección Intelectual.**
LPL **Ley de Protección Laboral.**
LPOO **Lenguajes de Programación Orientados a Objetos.**
LRU **Ley de Reforma Universitaria**
LS **Logical System Specification.**
LSA **Ley de Asociaciones Anónimas.**
LSB **Least Significant Bit.**
LSC **Large Scale Components.**
LSP **Liskov Substitution Principle.**
LTR **Lógica de Tiempo Real.**
MAN **Metropolitan Area Network.**
MAP **Ministerio para las Administraciones Públicas.**
MAPI **Messaging Application Program Interface.**
MBR **Master Boot Record.**
MCD **Modelo Conceptual de Datos.**
MCM **Method for Conceptual Modeling.**
MCTA **Modelo Conceptual de Tratamientos Analítico.**
MCVO **Modelo de Ciclo de Vida de los Objetos.**
MDA **Milestone Decision Authority.**
MENHIR **Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos.**
ME/R **Modelo Entidad/Interrelación.**
MFC **Microsoft Foundation Class library.**
MIB **Management Information Base.**
MIDAS **Multi-Tier Distributed Application Server.**
MIME **Multipurpose Internet Mail Extensions.**
MIPS **Millones de Instrucciones Por Segundo.**
MIS **Management Information System.**
MIT **Masschusetts Institute of Technology.**
MLD **Modelo Lógico de Datos.**
MLDR **Modelo Lógico de Datos Repartido.**
MLT **Modelo Lógico de Tratamientos.**
MLTR **Modelo Lógico de Tratamientos Repartido.**
MOD **Modelo Organizativo de Datos.**
MOI **Módulos de Ocultamiento de Información.**
MOM **Message Oriented Middleware.**
MOOD **Material's Object-Oriented Database.**
MOON **Méthode Orientée Objects Normalisés.**
MORE **Multimedia Oriented Repository Environment**.
MOSES **Methodology for Object-oriented Software Engineering of Systems.**
MOTA **Modelo Organizativo de Tratamientos Analítico.**
MPEG **Moving Pictures Experts Group.**
MRG **Modelo de Reutilización GIRO.**
MRP **Manufacturing Resource Planning.**
MS **Microsoft.**
MSB **Most Significative Bit.**
MST **Módulos de Sincronización de Tareas.**
MTA **Mail Transfer Agent.**
MVC **Model View Controller.**
MVE **Modular Visualisation Environment.**
MVS **Multiple Virtual Storage.**
NAG **Numerical Algorithms Group.**
NAS **Network Applications Support.**
NASA **National Aeronautics and Space Administration.**
NBS **National Bureau of Standards.**
NC **Network Computer.**
NCC **Norwegian Computer Center.**
NCOSE **National Council On Systems Engineering.**
NCS **Network Computing System.**
NCSA **National Center for Supercomputing Applications.**
NCSS **Non-Commented Source Statements.**
NDL **National Database Language.**
NFS **Network File System.**
NGPM **Next Generation Process Model.**
NHSE **National HPCC Software Exchange.**
NIAM **Nijssen Information Analysis Methodology.**
NICE **Non Profit Inernational Consortium for Eiffel.**
NIH **National Institute of Health.**
NIST **National Institute of Standards and Technology.**
NLH/E **Natural Language Help/English.**
NNTP **Network News Transfer Protocol.**
NOC **Number of Children.**
NOS **Network Operating System.**
NSA **National Security Agency.**
NSDIR **National Software Data and Information Repository.**
NT **New Technology.**
NTDFS **NT Distributed Filing System.**
NTOFS **NT Object Filing System.**
OAG **Open Application Group.**
OAI **Organización de Auditoria Informática.**
OASIS **Open and Active Specification of Information Systems**
OBA **Object Behavior Analysis.**
OCL **Object Constraint Language.**
OCP **Open-Closed Principle.**
OCR **Optical Character Readers.**
OCU **Object Connection Update.**
OCX **Object Linking and Embedding Custom Controls.**
ODA **Open Document Arquitecture.**
ODBC **Open Data Base Connectivity.**
ODM **Object Database Model.**
ODM **Organization Domain Modeling.**

ODMG **Object Database Management Group.**
ODMS **Object-oriented Database Management Systems.**
ODS **Object Description Skeleton.**
ODT **Object Definition Tool.**
OED **Oxford English Dictionary.**
OEM **Original Equipment Manufacturer.**
OLAP **On-Line Analytical Processing.**
OLE **Object Linking and Embedding.**
OLTP **On-Line Transaction Processing.**
OM **OASIS Method.**
OMA **Object Management Architecture.**
OMA **Object Mentor Associates.**
OMG **Object Management Group.**
OML **OPEN's Metamodel and Notation.**
OMT **Object Modeling Technique.**
ONC **Open Network Computing.**
OO **Object-Oriented.**
OO **Orientación a Objetos.**
OOA **Object-Oriented Analysis.**
OOAD **Object-Oriented Analysis and Design.**
OOAP **Object-Oriented Assistant Prototyper.**
OOCL **Object-Oriented Change and Learning.**
OOD **Object-Oriented Design.**
OODB **Object-Oriented Data Base.**
OODBMS **Object-Oriented Data Base Management System.**
OODCE **Object-Oriented Distributed Computing Environment.**
OODLE **Object-Oriented Design Language.**
OOIS **Object-Oriented Information Systems.**
OOL **Object-Oriented Language.**
OOP **Object-Oriented Programming.**
OOPSLA **Object-Oriented Programming Systems, Languages and Applications.**
OOSC **Object-Oriented Software Construction.**
OOSD **Object Oriented Structured Design.**
OOSE **Object Oriented Software Engineering.**
OOUI **Object Oriented User Interface.**
OOZE **Object Oriented Z Environment.**
OPEN **Object-oriented Process, Environment and Notation.**
OQL **Object Query Language.**
ORB **Object Request Broker.**
ORM **Object with Roles Model.**
OSA **Object-oriented Systems Analysis.**
OSD **Office of the Secretary of Defense.**
OSF **Open Software Foundation.**
OSI **Open System Interconnection.**
OT **Object Technology.**
OTAN **Organización del Tratado del Atlántico Norte.**
OTC **Object Technology Centers.**
OTUG **Object Technology Users Group.**
OUT **Object Under Test.**
OWL **ObjectWindows Library.**
PAL **Public Ada Library.**
PC **Personal Computer.**
PCT **Private Communication Technology.**
PCTE **Portable Common Tool Environment.**
PD **Patrón de Diseño.**
PD **Physical Design.**
PDF **Portable Document Format.**
PDL **Programs Design Lenguage.**
PDR **Preliminary Design Review.**
PERT **Program Evaluation and Review Technique.**
PF **Puntos de Función.**
PFA **Puntos de Función Ajustados.**
PFNA **Puntos de Función No Ajustados.**
PGGC **Plan General de Garantía de Calidad.**
PGP **Pretty Good Privacy.**
PGSR **Plan de Gestión y de Supervisión del Riesgo.**
PHS **Prosperity Heights Software.**
PIA **Modelo de Pruebas, Implantación y Aceptación del Sistema.**
PIAi **Actividad i del Modelo de Pruebas, Implantación y Aceptación del Sistema.**
PIE **Process Improvement Experiment.**
PIER **Process Improvement Experiment in Reuse.**
PII **Process Improvement Institute.**
PIN **Plan Informático Nacional.**
PIP **Process Improvement Paradigm.**
PIRS **Preliminary Interface Requirements Specification.**
PJ/NF **Projection-Join Normal Form.**
PLoP **Pattern Languages of Program Design.**
PLoPD **Pattern Languages of Program Design.**
PM **Program Manager.**
PMM **Process Maturity Model.**
PNG **Portable Network Graphics.**
POO **Programación Orientada a Objetos.**
POSS **Persistent Object Service Specification.**
PP **Proceso Primitivo.**
PPP **Point to Point Protocol.**
PROLOG **Programming in LOGic.**
PSL **Problems Specification Language.**
PSL/PSA **Problem Statement Language/Problem Statement Analyzer.**
PSRS **Preliminary Software Requirements Specification.**
PVVS **Plan de Verificación y Validación del Software.**
PYME **Pequeñas Y Medianas Empresas.**
QBE **Query By Example.**
QFD **Quality Function Deployment.**
QoS **Quality Of Service.**

QUEL **QUEry Language.**
QWAN **the Quality Without a Name.**
RA **Requirements Analysis.**
RAAT **Reuse Acquisition Action Team.**
RAD **Rapid Application Development.**
RAM **Random Access Memory.**
RAPPeL **Requirements-Analysis-Process Pattern Language for object-oriented development.**
RBSE **Repository Based Software Engineering.**
RD **Requirements Design.**
RDC **Revisión del Diseño Crítico.**
RDD **Responsability Driven Design.**
RdP **Red de Petri.**
RDP **Revisión del Diseño Preliminar.**
RDSI **Red Digital de Servicios Integrados.**
REBOOT **Reuse Based on Object-Oriented Techniques.**
RECAST **Requirements Composition And Specification Tool.**
RENOIR **European Requirements Engineering Network of Excellence.**
REP **Reuse/Release Equivalence Principle.**
RFC **Request For Comments.**
RFC **Response For a Class.**
RFP **Request For Proposals.**
RG **Revisión de Gestión.**
RIAT **Reuse Issues Action Team.**
RIB **Repository In a Box.**
RICIS **Research Institute for Computing and Information System.**
RID **Record ID.**
RIG **Reuse library Interoperability Group.**
RISC **Reduced Instruction Set Computer.**
RLF **Reuse Library Framework.**
RM **Reference Model.**
RMFF **Reuse Methodology Fusion Framework.**
RMI **Remote Method Invocation.**
ROAD **Report on Object Analysis & Design.**
ROADS **Reuse Oriented Approach for Domain based Software.**
ROI **Return Of Investment.**
ROM **Read Only Memory.**
ROOM **Real-Time Object-Oriented Modeling.**
ROSE **Reusable Object Software Engineering.**
ROSE **Reuse Of Software Elements.**
RP **Red de Petri.**
RPC **Remote Procedure Call.**
RPVVS **Revisión del Plan de Verificación y Validación del Software.**
RR.HH **Recursos Humanos.**
RRM **Reglamento Registro Mercantil.**
RRS **Revisión de Requisitos del Software.**
RS **Requirements Specification.**
RSC **Reuse Steering Committe.**
RSL **Reusable Software Library.**
RSOC **Rockwell Space Operations Company.**
RSRG **Reusable Software Research Group.**
RSVP **Rapid System Virtual Prototyping.**
RTB **Red Telefónica Básica.**
RTEE **Real Time Engineering Environment.**
RTF **Rich Text File.**
RTSA **Real-Time Structured Analysis.**
RTTI **Run Time Type Identification.**
RUP **Rational Unified Process.**
SA **Structured Analysis.**
SA/SD **Structured Analysis/Structured Design.**
SA-CMM **Software Acquisition Capability Maturity ModelSM.**
SADT **Structured Analysis and Design Technique.**
SAF **Specific Application Frame.**
SAIC **Science Applications International Corporation.**
SAP **Stable Abstractions Principle.**
SATC **Software Assurance Technology Center.**
SC **Subcomité.**
SCE **Software Capability Evaluation.**
SCE **Structure Chart Editor.**
SCSI **Small Computer System Interface.**
SDCCR **Software Development Capability/Capacity Review.**
SDCE **Software Development Capability Evaluation.**
SDD **Software Design Document.**
SDK **Software Development Kit.**
SDL **Software Development Library.**
SDLC **Systems Development Life Cycle.**
SDM **Semantic Data Model.**
SDP **Software Development Plan.**
SDP **Stable Dependencies Principle.**
SDRI **Sistema de Diccionario de Recursos de Información.**
SE **Sistemas Expertos.**
SE **Software Engineering.**
SEDDR **Software Engineering Data Definition and Representation.**
SEE **Software Engineering Environment.**
SEI **Software Engineering Institute.**
SEIS **Sociedad Española de Informática Sanitaria.**
SEN **Software Engineering Notes.**
SER **Software Evolution and Reuse.**
SERR **Software Engineering Risk Repository.**
SESC **Software Engineering Standards Committe.**
SET **Secure Electronic Transactions.**
SFA **Software Frameworks Association.**
SGBD **Sistema Gestor de Bases de Datos.**

SGBDR **Sistema Gestor de Bases de Datos Relacionales.**
SGBDRO **Sistema Gestor de Bases de Datos Objeto-Relacional.**
SGDRI **Sistema de Gestión de Diccionarios de Recursos de Información.**
SGI **Silicon Graphics Incorporated.**
SGL **Standard General Ledger.**
SGML **Standard Generalized Markup Language.**
SI **Sistema de Información.**
SIA **Sistema de Información Automatizado.**
SIB **Software Information Base.**
SICS **Swedish Institute for Computer Science.**
SIG **Sistemas de Información Geográfica.**
SIG **Special Interest Group.**
SIGMOD **Special Interest Group on Mangement Data.**
SLCSE **Software-Life Cycle Support Environment.**
SMS **Systems Management Server.**
SMTP **Simple Mail Transfer Protocol.**
SNAP **Semantic Net for Analysis & Prototyping.**
SNOBOL **StriNg Oriented symBOlic Language.**
SO **Sistema Operativo.**
SOA **Start Of Authority.**
SOHO **Small Office Home Office.**
SOM **IBM's System Object Model.**
SOM **Structured Object Method.**
SOMA **Semantic Object Modeling Approach.**
SOTR **Sistema Operativo en Tiempo Real.**
SPC **Software Productivity Centre.**
SPI **Software Process Improvement.**
SPQR **Software Productivity Quality and Reliability.**
SPR **Software Productivity Research.**
SQA **Software Quality Assurance.**
SQL **Structured Query Language.**
SRBM **Software Reuse Business Model.**
SREM **Software Requirements Engineering Methodology.**
SRS **Software Requirements Specifcation.**
SSADM **Structured System Analysis and Design Method.**
SSDD **System/Segment Design Document.**
SSR **Software Specification Review.**
SSR **Symposium on Software Reusability.**
SSS **System Segment Specification.**
SST **Sistemas Software Tradicionales.**
SST **Strategic Systems Technology.**
STARS **Software Technology Adaptable Reliable System.**
STC **Software Technology Conference.**
STL **Standard Templates Library.**
STP **Software Technology Program.**
STP **Software Test Plan.**
STR **Software Test Report.**
STSC **Software Technology Support Center.**
SUM **Software Usage Monitor.**
SURF **Software re-Use: a process impRovement experiment at an IBM Italia Facility.**
SWSC **Air Force Space and Warning System Centre.**
TAD **Tipo Abstracto de Datos.**
TAGS **Technology for the Automated Generation of Systems.**
TAPI **Telephony Application Programming Interface.**
TBD **To Be Determined.**
TCL/TK **Tool Command Language Toolkit.**
TCM **Toolkit for Conceptual Modeling.**
TCP/IP **Transmission Control Protocol/Internet Protocol.**
TDE **Transition Diagram Editor.**
TE **Tiempo Early.**
TI **Tecnologías de la Información.**
TL **Tiempo Late.**
TO **Tecnologías de Objetos.**
TO **Triggering Operation.**
TOA **The Object Agency.**
TODS **ACM Transactions On Database Systems.**
TOG **The Open Group.**
TP **Transaction Processing.**
TPI **Tamaño de Proceso de Información.**
TPV **Terminal Punto de Venta.**
TROOPER **The Reusable OO Parser for Eiffel Re-engineering.**
UAL **Unidad Aritmético Lógica.**
UBU **Universidad de Burgos.**
UC **Unidad de Control.**
UCP **Unidad Central de Proceso.`**
UDL **Unified Database Language.**
UE **Unión Europea.**
UIMS **User Interface Management Systems.**
UML **Unified Modeling Language.**
UoD **Universe of Discourse.**
UPM **Universidad Politécnica de Madrid.**
UPRR **Union Pacific RailRoad.**
UPV **Universidad Politécnica de Valencia.**
URI **Uniform Resource Identifier.**
URL **Uniform Resource Locator.**
USAL **Universidad de Salamanca.**
USD **United States Dollars.**
UVA **Universidad de Valladolid.**
UWA **User Work Area.**
V&V **Verificación y Validación.**
VB **Visual Basic.**
VBA **Visual Basic for Applications.**
VBX **Visual Basic eXtensions.**

VC++ **Visual C++.**
VCL **Visual Component Library.**
VDD **Version Description Document.**
VDL **Vienna Definition Language.**
VDM **Vienna Development Method.**
VHLL **Very High Level Languages**
VIFF **Visualization Image File Format.**
VM **Virtual Machine.**
VRML **Virtual Reality Modeling Language.**
W3C **World Wide Web Consortium.**
WAIS **Wide Area Information Servers.**
WAN **Wide Area Network.**
WBS **Working Breakdown Structure.**
WDM **Win32 Drivel Model.**
WEL **Windows Eiffel Library.**
WG **Working Group.**
WIDE **Workflow on Intelligent Distributed database Environment.**
WIMP **Windows, Icons, Menus and Pointing.**
WISR **Workshop on Institutionalizing Software Reuse.**
WISE **World Wide Information System.**
WMC **Weighted Methods per Class.**
WMRA **Write Many, Read Always.**
WORM **Write Once, Read Many Times.**
WSRD **Worldwide Software Resource Discovery.**
WWW **World Wide Web.**
WYSIWYG **What You See Is What You Get.**
XIE **X Image Extension.**
XML **Xperimental Markup Language.**
XPARC **Xerox Palo Alto Research Park.**
Y2K **Year 2000.**
YACC **Yet Another Compiler Compiler.**
YACL **Yet Another Class Library.**
YOOCC **Yes! An OO Compiler Compiler.**
YP **Yellow Pages.**
YSM **Yourdon Structured Method.**

Apéndice C
Glosario de Términos

A

Abstracción: Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes [Graham, 1994]. Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador [Booch, 1994].

Acoplamiento: Es el grado de interdependencia entre los módulos [Piattini et al., 1996].

Agregación: Forma especial de asociación que especifica una relación todo/parte entre el agregado (todo) y un componente (parte) [OMG, 1999].

Alfabeto: Se llama alfabeto a un conjunto finito y no vacío. Los elementos de un alfabeto se denominan letras o símbolos. Un alfabeto se representa por **Σ** [Alfonseca et al., 1990].

Algoritmo: Conjunto finito de reglas que dan una secuencia de operaciones para resolver un tipo específico de problema, o lo que es lo mismo, la traducción de aquél en una serie de instrucciones simples, en secuencia, encaminadas a realizar una tarea concreta.

Algoritmo Genético: Modelos computacionales basados en los mecanismos naturales de evolución genética de los organismos biológicos, que se aplican a la resolución de una amplia gama de problemas [García y Maudes, 1996].

Análisis: Distinción y separación de las partes de un todo hasta llegar a conocer sus principios o elementos [DRAE, 1995].

Análisis de Requisitos: Proceso de estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos [IEEE, 1999].

Análisis del Dominio: Actividad de identificar objetos y operaciones de un tipo de sistemas similares en un dominio del problema particular.

Análisis Orientado al Objeto: (AOO) Proceso que modela el dominio del problema identificando y especificando un conjunto de objetos semánticos que interaccionan y se comportan de acuerdo a los requisitos del sistema.

Árboles de Decisión: Modelo de una función discreta en la que se determina el valor de una variable y en función de su valor se lleva a cabo una acción.

Arquitectura del Software: Es la estructura global del software y las maneras en que esa estructura proporciona integridad conceptual a un sistema [Shaw and Garlan, 1995]. La estructura lógica y física de un sistema, forjada por todas las decisiones de diseño estratégicas y tácticas aplicadas durante el desarrollo [Booch, 1994]. La arquitectura del software se refiere a dos características importantes del software de computadora: la estructura jerárquica de los componentes procedimentales (*módulos*) y a la estructura de los datos [Pressman, 1992].

Arquitectura de Cuatro Capas: Es el framework conceptual de metamodelado generalmente aceptado. Explica las relaciones entre el meta metamodelo, el metamodelo, el modelo y el nivel de datos de usuario. Juntos forman las cuatro capas una encima de la otra [Metamodel, 1997].

Asociación: Una asociación describe un grupo de enlaces con una estructura y una semántica en común [Rumbaugh et al., 1991].Una relación que describe un conjunto de vínculos [OMG, 1999]. Una asociación representa una dependencia semántica entre clase e implica la dirección de esta dependencia [Joyanes, 1998].

Asset: Cualquier producto del ciclo de vida del software que pueda ser potencialmente reutilizado. Esto incluye: modelo de dominio, arquitectura de dominio, requisitos, diseño, código, bases de datos, esquemas de bases de datos, documentación, manuales de usuario, casos de prueba... [DoD, 1992], [NIST, 1994].

Atributo: Es un valor de un dato que está almacenado en los objetos de una clase [Rumbaugh et al., 1991]. Un atributo de una clase describe una información singular almacenada en cada instancia [Booch, 1994].

B

Bean: Es un componente software reutilizable que puede ser manipulado visualmente en una herramienta de desarrollo Java [Hamilton, 1997].

C

Calidad: Totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas [AENOR, 1992].

Calidad del Software: Conjunto global de características de un producto software, relacionado con su capacidad de satisfacer unas necesidades dadas; Grado en el que un software posee una combinación de atributos deseada; Grado en que el usuario percibe que el software satisface sus expectativas [AECC, 1986]. Grado con el que un sistema, componente o proceso cumple: *los requisitos especificados* y *las necesidades o expectativas del cliente o usuario* [IEEE, 1999].

Camino Crítico: Secuencia más larga de actividades conectadas a través de la red y que determina la duración total del proyecto [Piattini et al., 1996].

Campo Clave: Aquellos campos que identifican unívoca y mínimamente los registros.

Característica: Palabra genérica tanto para un atributo como para una operación [Rumbaugh et al., 1991].

CASE: Conjunto de herramientas y metodologías que soportan un enfoque de ingeniería en el desarrollo del software en alguna o en todas las fases de este proceso [Piattini et al., 1996].

Caso de Uso: Forma o patrón o ejemplo concreto de utilización, un escenario que comienza con algún usuario del sistema que inicia alguna transacción o secuencia de eventos interrelacionados [Jacobson et al., 1993].

Centro de Transformación: Es la parte del DFD que contiene las funciones esenciales del sistema, independientemente de la implementación particular de la entrada y de la salida [Piattini et al., 1996].

Cibernética: Ciencia que estudia comparativamente los sistemas de comunicación y regulación automática de los seres vivos con sistemas electrónicos y mecánicos semejantes a aquellos. Entre sus aplicaciones está el arte de construir y manejar aparatos y máquinas que mediante procedimientos electrónicos efectúan automáticamente cálculos complicados y otras operaciones similares [DRAE, 1995].

Ciclo de Desarrollo del Software: Se denomina ciclo de desarrollo del software al período de tiempo que comienza con la decisión de desarrollar un producto software y finaliza cuando se ha entregado éste. Este ciclo incluye, en general, una fase de requisitos, una fase de diseño, una fase de implantación, una fase de pruebas, y a veces, una fase de instalación y aceptación [AECC, 1986].

Ciclo de Vida del Software: Es un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software,

abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso [ISO/IEC, 1995].

Clase: Es un conjunto de objetos que comparten una estructura común y un comportamiento común [Booch, 1994]. Es la implementación de un tipo de objeto (*considerando los objetos como instancias de las clases*) [Piattini et al., 1996].

Clase Abstracta: Clase que no representa completamente un objeto. En su lugar, representa un amplio rango de diferentes clases de objetos. Sin embargo, esta representación mantiene sólo las características que dichas clases de objetos tienen en común. Por lo tanto, una clase abstracta sólo ofrece una descripción parcial de sus objetos [Martin, 1992]. Una clase que no tiene instancias. Una clase abstracta se escribe con la intención de que sus subclases concretas añadan elementos nuevos a su estructura y comportamiento, normalmente implementando sus operaciones abstractas [Booch, 1994]. Clase que no puede tener instancias directas pero cuyos descendientes sí pueden tenerlas [Rumbaugh et al., 1991].

Clase Concreta: Clase que puede tener instancias directas [Rumbaugh et al., 1991].

Clase Contenedora: Clase de objetos contenedores. Entre los ejemplos se incluyen los conjuntos, las matrices, los diccionarios y las asociaciones [Rumbaugh, et al., 1991].

Clase Genérica: Véase clase parametrizada.

Clase Parametrizada: Plantilla para la creación de clases reales que pueden diferir en formas bien definidas, como indican sus parámetros en el momento de creación. A menudo, los parámetros son tipos de datos o clases, pero pueden incluir otros atributos, como el tamaño de una colección. También denominada clase genérica [Rumbaugh et al, 1991].

Cohesión: Medida de la relación funcional de los elementos de un módulo [Piattini et al., 1996].

Complejidad Ciclomática: Métrica del software que proporciona una medición cuantitativa de la complejidad lógica de una programa [Pressman, 1997].

Componente: Un componente es una unidad de composición con unas interfaces contractualmente especificadas con dependencias exclusivamente del contexto. Los componentes pueden ser desarrollados de forma independiente y son integrados por terceras partes [Szyperski and Pfister, 1996].

Componentware: Estrategia de desarrollo que pretende, en lugar de construir grandes aplicaciones de forma completa, dedicarse a la creación de unidades arquitectónicas, denominadas componentes, con el objetivo de integrarlas para, que colaborando entre sí, obtener una funcionalidad mucho mayor que la que cada uno de ellos puede ofrecer por separado.

Comportamiento de un objeto: El comportamiento de un objeto es cómo actúa y reacciona un objeto, en función de sus cambios de estado y paso de mensajes [Booch, 1994].

Composición: Relación semántica que describe una forma de agregación con un matiz fuerte de propiedad y de coincidencia de vida como partes de un todo. Las partes con una multiplicidad que no es fija pueden ser creadas después del objeto compuesto, pero una vez creadas ellas viven y mueren con él. Estas partes pueden ser también eliminadas antes de la muerte del objeto compuesto. La composición puede ser recursiva [OMG, 1999].

Concurrencia: (En OO) Es la propiedad que distingue un objeto activo de uno que no está activo [Booch, 1994].

Control de Proyecto: Consiste en la supervisión periódica y en la comparación de los resultados con los previstos en el calendario [Piattini et al., 1996].

Covarianza: Propiedad por la que cuando en un proceso de especialización, se produce una redefinición de un atributo en la subclase, el atributo de la superclase y el atributo redefinido deben variar juntos [Meyer, 1997].

D

Dato: Están constituidos por registros de los hechos, acontecimientos, transacciones... [Lucey, 1991].

Deliverable: Véase producto a entregar.

Descomposición del Trabajo: Técnica que permite representar las actividades que hay que realizar a distinto nivel de detalle por medio de un diagrama de estructuras. Se corresponde con el término **Working Breakdown Structure (WBS)** [Piattini et al., 1996].

Diagrama de Clases: Es el diagrama que muestra una colección de elementos declarativos (estáticos) del modelo, como clases y tipos, sus contenidos y relaciones [OMG, 1999].

Diagrama de Colaboraciones: Es un diagrama que muestra interacciones entre objetos organizados alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero de una forma diferente [OMG,1999].

Diagrama de Contexto: El diagrama de contexto, o diagrama de nivel 0, es el primero de la jerarquía de DFD. El objetivo de este diagrama es delimitar la frontera entre el sistema con el mundo exterior y definir sus interfaces, esto es, los flujos de entrada y salida del sistema con el entorno. El diagrama de contexto está formado por un proceso que se representa como una **caja negra** del sistema completo, un conjunto de entidades externas que presentan la procedencia y el destino de la información del sistema, y un conjunto de flujos de datos que representan los caminos por los que fluye dicha información [Piattini et al., 1996].

Diagrama de Estructuras: Técnica que permite definir cuándo, bajo qué condiciones y cuántas veces se tienen que realizar los tratamientos identificados en los procesos de un DFD [MAP, 1995].

Diagrama de Estructura de Cuadros de Constantine: Nombre que recibe en Métrica 2.1 el *diagrama de estructuras*. Véase Diagramas de Estructuras. [MAP, 1995].

Diagrama de Estructura de Datos: Modelo de información compuesto exclusivamente por relaciones **1:N** [Piattini et al., 1996].

Diagrama de Flujo de Datos: DFD. Es un diagrama en forma de red que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema [Piattini et al., 1996].

Diagrama de Instancias: Describe la forma en que un cierto conjunto de objetos se relacionan entre sí. Describe instancias de objetos [Rumbaugh et al., 1991]. Este concepto se corresponde con el concepto de diagrama de objetos de UML [OMG, 1999].

Diagrama de Objetos: Los diagramas de objetos proporcionan una notación gráfica formal para el modelado de objetos, clases y sus relaciones entre sí, son útiles, tanto para el modelado abstracto como, para diseñar programas reales [Rumbaugh et al., 1991]. Diagrama que contiene objetos y sus relaciones en un momento dado del tiempo. Un diagrama de objetos puede ser considerado un caso especial de un diagrama de clases o de un diagrama de colaboraciones [OMG, 1999]. Esta definición se ajusta al concepto de diagrama de instancias de Rumbaugh en OMT [Rumbaugh et al., 1991].

Diagrama de Secuencias: Un diagrama que muestra interacciones entre objetos organizadas en secuencia temporal. En particular muestra los objetos participantes de la interacción y la secuencia de mensajes intercambiados. A diferencia del diagrama de colaboraciones, un diagrama de secuencias incluye la secuencia temporal pero no incluye las relaciones entre los objetos. Pueden existir diagramas de secuencias genéricos (que describen todos los escenarios posibles) y de instancias (que describen un escenario particular). Los diagramas de secuencias y de colaboraciones expresan información similar pero de una manera diferente [OMG, 1999].

Diagrama de Transición de Estados: DTE. Diagrama que representa los estados que puede tomar un componente o un sistema y que, además, muestra los eventos o circunstancias que implican el cambio de un estado a otro.

Diagrama Estático de Estructuras: Los diagramas estáticos de estructura muestran la estructura estática del modelo; en concreto muestran las cosas que existen como son las clases y tipos, su estructura interna, y su relación con el resto de los elementos [OMG, 1999]. Este concepto se corresponde con el concepto de diagrama de objetos enunciado por Rumbaugh para OMT [Rumbaugh et al., 1991].

Diccionario de Datos: DD. Lista organizada de todos los datos utilizados por el sistema, con definiciones precisas y rigurosas, para que cliente y analista tengan una visión común de todos los flujos y almacenes.

Diccionario de Recursos de Información: DRI o en inglés – ***IRD*** *Information Resource Dictionary*. Depósito integrado de todos los datos sobre la organización, automatizados o no, que son utilizados para efectuar las labores de planificación, control y operación que permitan a la empresa cumplir sus objetivos [Piattini et al., 1996].

Diseño Software: Es el proceso de definición de la arquitectura software: componentes módulos, interfaces, procedimientos de prueba y datos de un sistema que se crean para satisfacer unos requisitos especificados.

Diseño Arquitectónico: Define las relaciones entre los principales elementos estructurales del programa [Pressman, 1997].

Diseño de Datos: El diseño de datos transforma el modelo de dominio de la información, creado durante el análisis, en las estructuras de datos necesarias para la implementación del software [Pressman, 1997].

Diseño de la Interfaz: El diseño de la interfaz describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean [Pressman, 1997].

Diseño Estructurado: Es el arte de diseñar los componentes de un sistema y la relación entre ellos de la mejor forma posible [Yourdon and Constantine, 1979]. Es el proceso de decidir la forma en la cual componentes interconectados resolverán un problema bien especificado [Yourdon and Constantine, 1979].

Diseño Orientado al Objeto: (DOO) Proceso que modela el dominio de la solución, lo que incluye a las clases semánticas con posibles añadidos, y las clases de interfaz, aplicación y utilidad identificadas durante el diseño.

Diseño Procedimental: El diseño procedimental transforma elementos estructurales de la arquitectura del programa en una descripción procedimental de los componentes del software [Pressman, 1997].

Director de Proyecto: Persona que tiene la responsabilidad de planificar, controlar y dirigir las actividades del proyecto. Muchas veces estas responsabilidades suponen la coordinación e integración de actividades a través de las unidades organizativas [Piattini et al., 1996].

Documento Base: Producto que ha sido formalmente revisado y aceptado, y que sólo puede cambiar mediante un procedimiento formal de control de cambios. Esto se realiza por medio de una revisión formal del grupo de control de cambios. El término equivalente en la bibliografía en inglés es *baseline* [Piattini et al., 1996].

E

Encapsulamiento: Es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar la interfaz contractual de una abstracción y su implantación [Booch, 1994]. Es un principio de estado que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto [Piattini et al., 1996].

Enfoque Sistemático: Es la manera de estudiar o analizar sistemas adoptando una visión global de los mismos, que se va refinando progresivamente mediante una descomposición de arriba abajo [Piattini et al., 1996].

Enlace: Conexión física o conceptual entre instancias de objetos [Rumbaugh et al., 1991].

Entidad: Objeto real o abstracto acerca del cual se quiere almacenar información en una base de datos.

Envoltorio: Véase *wrapper*.

Ergonomía: Estudio de datos biológicos y tecnológicos aplicados a problemas de mutua adaptación entre el hombre y la máquina.

Especificación: Es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño, el comportamiento u otras características de un sistema o componente de un sistema [IEEE, 1999].

Especificación Formal: Una especificación formal sería la que se describe mediante sintaxis y semánticas formales con las que se especifica el funcionamiento y el comportamiento del sistema. Una especificación formal a menudo tiene una forma matemática (*por ejemplo, se puede utilizar el cálculo de predicados como base de un lenguaje de especificación formal*) [Pressman, 1992].

Estado: Conjunto de circunstancias o atributos que caracterizan a una persona o cosa en un tiempo dado.

Estado de un objeto: El estado de un objeto abarca todas las propiedades (*normalmente estáticas*) del mismo más los valores actuales (*normalmente dinámicos*) de cada una de esas propiedades [Booch, 1994].

Estructura de Datos: Es una representación de la relación lógica existente entre los elementos individuales de datos [Pressman, 1997].

Estructura del Programa: Ver jerarquía de control.

Excepción: Una excepción es una señal especial que puede ser activada por una cierta instrucción y manejada por otra, posiblemente en una parte remota del sistema [Meyer, 1997].

F

Forma de Utilización: Véase caso de uso.

Función Primitiva: Las funciones o procesos primitivos son aquellos procesos de un DFD que no se descomponen en diagramas de nivel inferior. Por cada función primitiva tiene que existir una especificación que la describa [Piattini et al., 1996].

G

Generalización: Es una relación de clasificación entre un elemento más general y un elemento más específico. El elemento más específico es completamente consistente con el elemento más general, conteniendo información adicional.

Gestión de la Configuración: Disciplina de coordinar el desarrollo del software y controlar el cambio y evolución de los productos software y de sus componentes [Schmerl, 1996].

Gestión de Proyectos: Proceso de planificar, organizar, asignar personal, dirigir y controlar la producción de un sistema [CERN, 1997].

Gestión de Riesgos: Disciplina cuyos objetivos son identificar, direccionar y eliminar los elementos de riesgo para el software antes de que consigan entorpecer el éxito del software o convertirse en una fuente de trabajo software (*trabajo ya realizado anteriormente de otras formas*) más caro [Boehm, 1989].

H

Herencia: Mecanismo por el que elementos más específicos incorporan estructura y comportamiento de elementos más generales relacionados por el comportamiento [OMG, 1999]. Relación entre clases, en la que una clase comparte la estructura o comportamiento definido en otra (*herencia simple*) u otras (*herencia múltiple*) clases. La herencia define una relación "*de tipo*" entre clases en la que una subclase hereda de una o más superclases generalizadas; una subclase suele especializar a sus superclases aumentando o redefiniendo la estructura y comportamiento existentes [Booch, 1994].

Herencia de Constantes: Tipo de herencia de propiedades en la que todas las propiedades de la superclase **A** son constantes que describen objetos compartidos [Meyer, 1997].

Herencia de Conversión a Diferida: Tipo de herencia con variación. Se aplica cuando **B** redefine algunas características concretas de **A** en características abstractas [Meyer, 1997].

Herencia de Extensión: Perteneciente a la familia de herencia de modelado. Se aplica cuando la subclase **B** introduce características no presentes en la superclase **A** y no aplicables a las diferentes instancias de **A**. La clase **A** debe ser concreta [Meyer, 1997].

Herencia de Implementación: Perteneciente a la familia de herencia software. Herencia estructural que se aplica si una subclase **B** obtiene de una superclase **A** un conjunto de características (*otras que atributos constantes y funciones*) necesarias para la implementación de la abstracción asociada con **B**. Ambas clases **A** y **B** deben ser concretas [Meyer, 1997].

Herencia de Máquina: Tipo de herencia de propiedades en la que todas las propiedades de la superclase **A** son rutinas que pueden verse como operaciones de una máquina abstracta [Meyer, 1997].

Herencia de Modelado: Refleja relaciones "es un" entre las abstracciones de un modelo [Meyer, 1997].

Herencia de Propiedades: Perteneciente a la familia de herencia software. Se aplica cuando una superclase **A** existe solamente con el propósito de ofrecer un conjunto de propiedades relacionadas lógicamente para el beneficio de sus subclases como **B**. Existen dos variantes: ***herencia de constantes*** y ***herencia de máquina*** [Meyer, 1997].

Herencia de Reificación: (o de materialización) Perteneciente a la familia de herencia de software. Se aplica si una superclase **A** representa un tipo general de estructura de datos, y la subclase **B** representa una elección de implementación parcial o completa para las estructuras de datos de ese tipo. **A** es abstracta; **B**

puede ser abstracta, dejando espacio para posteriores reificaciones a través de sus propias subclases, o puede ser concreta [Meyer, 1997].

Herencia de Restricción: Perteneciente a la familia de herencia de modelado. Se aplica si las instancias de **B** son aquellas instancias de **A** que cumplen una cierta restricción, expresada si es posible como parte de la invariante de **B** y no incluida en la invariante de **A**. Cualquier característica introducida por **B** debe ser una consecuencia lógica de añadir una restricción. **A** y **B** deben ser ambas abstractas o ambas concretas [Meyer, 1997].

Herencia de Variación: Sirve para describir una clase mediante sus diferencias con otras clases [Meyer, 1997]. La herencia con variación se aplica cuando **B** redefine algunas características de **A**; **A** y **B** son ambas abstractas o ambas concretas, y **B** no puede introducir más características excepto las necesarias para redefinir las características. Existen dos casos herencia con variación funcional y herencia con variación de tipo [Meyer, 1997].

Herencia de Variación de Tipo: Herencia con variación, donde todas las redefiniciones son redefiniciones de las signaturas de las operaciones [Meyer, 1997].

Herencia de Variación Funcional: Herencia con variación, donde las redefiniciones afectan al cuerpo de las implementaciones operaciones, en lugar de a las signaturas de las mismas [Meyer, 1997].

Herencia de Vistas: Perteneciente a la familia de herencia de modelado. Varios descendentes de una cierta clase representan conjuntos de instancias no disjuntos debido a la existencia de varias formas de clasificar las instancias del padre [Meyer, 1997].

Herencia Estructural: Perteneciente a la familia de herencia software. Se aplica si una superclase **A**, una clase abstracta, representa una propiedad estructural y una subclase **B**, que puede ser abstracta o concreta, representa un cierto tipo de objetos que poseen dicha propiedad [Meyer, 1997].

Herencia Múltiple: Tipo de herencia que permite a una clase tener más de una superclase y heredar características de sus ancestros [Rumbaugh et al., 1991].

Herencia por Subtipado: Perteneciente a la familia de herencia de modelado. La herencia por subtipado se aplica si **A** y **B** representan ciertos conjuntos **A'** y **B'** de objetos externos, de forma que **B'** es un subconjunto de **A'** y el conjunto modelado por otro cualquier subtipo de herencia de **A** es disjunto de **B'**. **A** debe ser abstracta [Meyer, 1997].

Herencia Simple: Tipo de herencia por la que una clase sólo puede tener una superclase [Rumbaugh et al., 1991].

Herencia Software: Expresa relaciones software, que no son obvias en el modelo [Meyer, 1997].

Hito: Del inglés *milestone*. Representa un suceso o evento en el tiempo del que algún miembro del proyecto se hace responsable y que se utiliza para medir el progreso [Piattini et al., 1996].

I

Identidad: Es aquella propiedad de un objeto que lo distingue de todos los demás [Booch, 1994].

Información: Dato o conjunto de datos que, en un contexto determinado, tienen un significado para alguien y transmiten un mensaje útil en un lugar determinado [Monforte, 1995].

Ingeniería del Software: Es la aplicación de herramientas, métodos y procedimientos de forma eficiente en cuanto al coste para producir y mantener una solución a un problema de procesamiento real, automatizándolo parcial o totalmente mediante el software [Horan, 1995].

Ingeniería Inversa: La ingeniería inversa del software es el proceso consistente en analizar un programa en un esfuerzo por crear una representación del programa con un nivel de abstracción más elevado que el código fuente [Pressman, 1997].

Instanciación: Proceso de creación de instancias de clases [Rumbaugh et al., 1991].

Interrelación: Es aquella asociación o correspondencia existente entre entidades.

J

Jerarquía: Es una clasificación u ordenación de abstracciones [Booch, 1994].

Jerarquía de Control: Representa la organización (a menudo jerárquica) de componentes del programa (módulos) e implica una jerarquía de control [Pressman, 1997].

L

Lenguaje: Lenguaje sobre el alfabeto **Σ** es todo subconjunto del lenguaje universal de **Σ** [Alfonseca et al., 1990].

Lenguaje Estructurado: Lenguaje de especificación que hace uso de un vocabulario y una sintaxis limitados.

Lenguaje Formal: Es un lenguaje compuesto de tres componentes principales: una sintaxis, que define la notación específica con la que se representa la especificación; una semántica, que ayuda a definir un universo de objetos que se usarán para describir el sistema, y por último, un conjunto de relaciones que definen las reglas que indican qué objetos satisfacen adecuadamente la especificación [Pressman, 1992].

Lenguaje Universal: Es el conjunto de todas las palabras que se pueden formar de un alfabeto **Σ**. También se denomina ***universo del discurso***, y se representa por **W(Σ)** o Σ^*. El lenguaje universal es un conjunto infinito [Alfonseca et al., 1990].

Ligadura: Ligadura es el proceso por el que se vincula el nombre del servicio a la implementación. En general el sistema selecciona el método más específico [Piattini, 1996].

M

Matriz Entidad/Función: Visualiza las relaciones existentes entre las funciones que lleva a cabo un sistema y la información necesaria para soportar las mismas [Piattini et al., 1996].

Mensaje: Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento [OMG, 1999].

Meta metamodelo: Un meta metamodelo define el lenguaje para expresar metamodelos [Metamodel, 1997].

Metaclase: Una clase cuyas instancias son a su vez clases [Berard, 1996].

Metadatos: Datos acerca de los datos. Los metadatos que están presentes durante la ejecución permiten a la aplicación razonar acerca de su propia estructura y capacidades, con posibilidad cambiarlas; aquí se incluyen las operaciones que admiten los objetos, los atributos que poseen o los tipos de los atributos.

Metamodelo: El modelo de información para la información que puede ser expresada durante el modelado [Metamodel, 1997]. Un modelo que define el lenguaje para expresar un modelo [OMG, 1999].

Método: Es la implementación de una operación en una clase. El algoritmo o procedimiento que permite llegar al resultado de una operación [OMG, 1999].

Metodología: Una metodología de Ingeniería del Software es un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas [Rumbaugh et al., 1991].

Milestone: Véase hito.

Modelo: Abstracción de un sistema semánticamente cerrada [OMG, 1999]. Una colección de elementos ensamblados durante el modelado de un sistema, como puede ser un sistema software [Metamodel, 1997].

Modelo de Componentes Software: Es una especificación de cómo desarrollar componentes software reutilizables, y cómo estos objetos pueden comunicarse con el resto.

Modelo de Objetos: La colección de principios que forman las bases del diseño orientado a objetos; un paradigma de Ingeniería del Software que enfatiza los principios de abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia [Booch, 1994].

Modelo de Proceso Software: Formalismos que permiten la representación de los conceptos que subyacen al proceso software.

Modelo Esencial: El modelo esencial del sistema es un modelo de lo que el sistema debe hacer para satisfacer los requisitos del usuario, diciendo lo menos posible (*preferiblemente nada*) acerca de cómo se implantará [Yourdon, 1989].

Modelo Semántico: Modelo que captura el significado de las entidades y las relaciones del mundo real.

Modularidad: Es el atributo individual del software que permite a un programa ser intelectualmente manejable [Myers, 1978]. Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados [Booch, 1994].

Módulo: Parte lógica separable de un programa [AECC, 1986]. Subconjunto coherente de un subsistema que contiene un grupo de clases fuertemente acotadas y sus interrelaciones [Rumbaugh et al., 1991].

Multiplicidad: Especifica el número de instancias de una clase que pueden estar relacionadas con una única instancia de una clase asociada [Rumbaugh et al., 1991].

N

Normalización: Reducción sucesiva de un conjunto dado de relaciones a una forma más deseable.

O

Objetivo de Proyecto: Enunciado que especifica los resultados a conseguir. Las características que debe cumplir un objetivo de proyecto para que quede bien definido son: **Asequible**: Meta que se puede alcanzar dentro de un tiempo determinado y con unas restricciones dadas. **Definitivo**: Especifica qué es lo que se tiene que conseguir de forma concreta. **Cuantificable**: Especifica un criterio de finalización.**De duración específica**: Define la duración de las actividades. [Piattini et al., 1996].

Objeto: En el ámbito conceptual un objeto es una entidad percibida en el sistema que se está desarrollando, mientras que al nivel de implementación, un objeto se corresponde con un encapsulamiento de un conjunto de operaciones (servicios) que pueden ser invocadas externamente y de un estado que recuerda el efecto de sus servicios. Un objeto se describe por sus propiedades, también llamadas atributos (*estructura del objeto*) y por los servicios que puede proporcionar (*comportamiento del objeto*). El estado de un objeto viene determinado por los valores que toman sus atributos, valores que han de cumplir siempre las restricciones impuestas sobre ellos [Piattini et al., 1996]. Algo a lo cual se le puede hacer algo. Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares se definen en su clase común. Los términos instancia y objeto son intercambiables [Booch, 1994].

Objeto Activo: Aquel objeto extraído de una abstracción del mundo real que representa un hilo separado de control (*una abstracción de un proceso*) [Booch, 1994].

Operación: Es una función o transformación que se puede aplicar o que puede ser aplicada por los objetos de una clase [Rumbaugh et al., 1991].

P

Palabra: Se denomina *palabra formada con los símbolos de un alfabeto*, a toda secuencia finita de letras de ese alfabeto [Alfonseca et al., 1990].

Paradigma: (Del latín *paradigma*, del griego *paradeigma*) El significado original era el de ejemplo ilustrativo, en particular un enunciado modelo que muestra todas las reflexiones de una palabra. Sin embargo, el historiador Thomas Kuhn en su libro *The Structure of Scientific Revolutions* (La estructura de las revoluciones científicas) extendió la definición de la palabra para abarcar un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, esto es, una forma de ver el mundo [Kuhn, 1971].

Persistencia: Es la propiedad de un objeto por la que su existencia trasciende el tiempo (*es decir, el objeto continúa existiendo después de que su creador deja de existir*) y/o el espacio (*es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado*) [Booch, 1994].

Plan de Proyecto: Un documento de gestión que describe la realización del proyecto. El plan describe, entre otros, el trabajo que ha de llevarse a cabo, los recursos necesarios y los métodos a utilizar [AECC, 1986].

Polimorfismo: La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases [Graham, 1994]. Concepto de la teoría de tipos, de acuerdo con el cual un nombre (como una declaración de una variable) puede denotar objetos de muchas clases diferentes que se relacionan mediante alguna superclase

común; así todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes modos [Booch, 1994]

Política de Calidad: Directrices y objetivos generales de una Institución relativos a la calidad, expresados formalmente por la Dirección General.

Powertype: Es un clasificador estereotipado que denota que el clasificador es un metatipo, cuyas instancias son subtipo de otro tipo [OMG, 1999]. Es una dependencia estereotipada cuya fuente es un conjunto de generalizaciones y cuyo destino es un clasificador que especifica que el destino es el *powertype* de la fuente [OMG, 1999].

Proceso Ligero: Suele existir dentro de un solo proceso del sistema operativo en compañía de otros procesos ligeros, que comparten el mismo espacio de direcciones [Booch, 1994].

Proceso Pesado: Aquel típicamente manejado de forma independiente por el sistema operativo de destino, y abarca su propio espacio de direcciones [Booch, 1994].

Producto a entregar: Del inglés *deliverable*. Es un producto o servicio final que especifica el cliente o usuario y que se le entrega al final del proyecto [Piattini et al., 1996].

Proceso Software: El conjunto de actividades necesarias para transformar las ideas iniciales del usuario que desea automatizar un determinado trabajo en software.

Programación Orientada a Objetos: POO. La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia [Booch, 1994].

Protocolo: (*En orientación al objeto*) Al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las formas de invocación u órdenes que admite, se le denomina su protocolo. Un protocolo denota las formas en las que un objeto puede actuar y reaccionar, y de esta forma constituye la visión externa completa, estática y dinámica, de la abstracción [Booch, 1994].

Prototipo: Sistema software que incluye sólo algunas características del sistema final con objeto de mejorar la comprensión del diseño o de su realización [AECC, 1986].

Prototipo Robusto: Prototipo que puede evolucionar hacia un sistema final.

Proyecto: Conjunto de etapas, actividades y tareas para alcanzar un objetivo que implica un trabajo no inmediato a un plazo relativamente largo [Piattini et al., 1996].

Puntos de Función: Es una medida del tamaño de las aplicaciones de ordenador y de los proyectos donde se construyen éstas. El tamaño es medido desde un punto de vista funcional, o de usuario. Es independiente del lenguaje de programación, de la metodología de desarrollo, de la tecnología o de la capacidad del equipo del proyecto [Boehm, 1996].

R

RAD: (*Rapid Application Development*) Los entornos de desarrollo rápido de aplicaciones se basan en la creación de las aplicaciones alrededor de formularios, que no son más que ventanas inicialmente vacías. En los formularios se van colocando los controles necesarios, los cuales tienen una serie de propiedades asociadas cuyos valores son definidos a la hora de crear el programa.

Red de Petri: Una red de Petri es un grafo orientado formado por plazas o lugares, representados por círculos, transiciones, representadas por segmentos rectilíneos, y por un conjunto de arcos dirigidos que unen ambos.

Red Neuronal: Es una red de varios procesadores simples, cada uno de los cuales puede tener una pequeña cantidad de memoria local. Estas unidades están conectadas por canales de comunicación los cuales normalmente transportan datos numéricos codificados de diferentes formas. Estas unidades operan sólo con sus datos locales sobre las entradas que reciben por sus conexiones de entrada [Sarle, 1996].

Redes de Precedencia: Las redes de precedencia constituyen una representación gráfica del proyecto donde se relacionan las actividades de tal forma que se pueden visualizar las que son críticas. Además, se pueden representar en una escala de tiempos para facilitar la distribución de recursos y la determinación del presupuesto [Piattini et al., 1996].

Refinamiento: Es la descomposición funcional del sistema mediante refinamientos sucesivos a diferentes niveles de abstracción.

Regla de Negocio: Una sentencia que define o restringe algunos aspectos del negocio. Es un intento de recoger la estructura del negocio o de controlar o influenciar el comportamiento del negocio.

Reificación: Proceso de hacer que conceptos externos estén disponibles en el nivel objeto [Madany et al., 1991]. El proceso de considerar algo abstracto en una entidad material [Web, 1997].

Reingeniería: Transformación sistemática de un sistema existente en una nueva forma que realice mejoras de calidad en las operaciones, capacidades del sistema, funcionalidades de rendimiento o capacidad de evolucionar a un menor coste, calendario o riesgo para el cliente [Tilley et al., 1995].

Relación semántica: Una conexión semántica entre elementos de un modelo [OMG,1999].

Repositorio: Tipo de diccionario que contienen las herramientas **CASE** donde se almacenan los datos generados durante el ciclo de vida de un desarrollo: *esquemas, grafos, matrices, información relativa a la gestión de proyectos…* Existen dos tipos de repositorios. Los **r**epositorios (*con r minúscula*) que almacenan los objetos de una herramienta **CASE** particular. Los **R**epositorios (*con R mayúscula*), que tienen un alcance mayor, se basan en estándares e implementan un modelo de información abierto y extensible, soportando un entorno integrado de ingeniería del software [Piattini et al., 1996].

Requisito: Condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado [Piattini et al., 1996].

Reutilización: Utilización de conceptos y objetos existentes en un sistema o situación nueva, directamente o adaptándolos. Para ello, estos conceptos y objetos deberán encontrarse codificados en un nivel de abstracción establecido y deberán poder ser recuperados [Krueger, 1992]. Cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior [Freeman, 1987].

Riesgo: Circunstancias potencialmente adversas que pueden afectar a un proceso de desarrollo o a la calidad de los productos [Ghezzi et al., 1991].

S

Sistema: Conjunto ordenado de cosas que, ordenadamente relacionadas entre sí, contribuyen a un determinado objetivo [DRAE, 1995].

Sistema de Información: Un conjunto formal de procesos que, operando sobre una colección de datos estructurada según las necesidades de la empresa, recopilan, elaboran y distribuyen la información (*o parte de ella*) necesaria para las operaciones de dicha empresa y para las actividades de dirección y control correspondientes (*decisiones*) para desempeñar su actividad de acuerdo a su estrategia de negocio [Andreu et al., 1996].

Software: Programas, procedimientos, reglas y la posible documentación asociada y datos que pertenezcan a la explotación de un sistema de ordenador [AECC, 1986].

Software de Tiempo Real: Sistema que controla un ambiente o un entorno determinado. El sistema recoge datos de este entorno, los procesa o analiza y produce una salida con suficiente rapidez como para influir en ese entorno [Pressman, 1992].

Structure Chart: Véase Diagrama de Estructuras.

T

Tecnologías de la Información: (**TI**) Término para hacer referencia a la informática, las tecnologías de comunicación y cualquier otra técnica que permita manejar, comunicar y procesar la información en cualquiera de los formatos en que pueda presentarse [Piattini et al., 1996].

Telos: Lenguaje de representación que soporta una especificación de clases multinivel y atributos de clase, herencia e instanciación. Se utiliza como lenguaje de especificación en el **SIB** de **ITHACA**.

Término Local: Es aquél que se define explícitamente en una especificación de proceso individual [Yourdon, 1989].

Tiempo de vida de un objeto: El tiempo de vida de un objeto es el tiempo que se extiende desde el momento en que se crea un objeto por primera vez (consumiendo así espacio por primera vez) hasta que ese espacio se recupera [Booch, 1994].

Tipo Abstracto de Datos: TAD o **ADT**. Un tipo abstracto de datos es una colección de valores y de operaciones que se definen mediante una especificación que es independiente de cualquier representación [Peña, 1998].

U

Uso: Relación semántica que supone un refinamiento de una asociación, por el que se establece qué abstracción es el cliente y qué abstracción es el servidor que proporciona ciertos servicios [OMG, 1999].

V

Versión: Una versión de un objeto es una instantánea semánticamente significativa del objeto, tomada en un momento dado en el tiempo [Bertino, 1993].

Viabilidad Legal: Comprobación de que los requisitos del sistema no vayan contra alguna ley o reglamento (por ejemplo, la LORTAD) o a disposiciones legales de contratos, responsabilidad civil… [Piattini et al., 1996].

W

Working Breakdown Structure: WBS – Véase Descomposición del trabajo.

Wrapper: Clase u operación que envuelve o encapsula llamadas a rutinas de biblioteca o cualquier otro código que está siendo reutilizado. Sinónimo de envoltorio [Rumbaugh et al., 1991].

C.1 Referencias

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[AENOR, 1992] AENOR. *"Normas para la Gestión y el Aseguramiento de la Calidad*". AENOR, 1992.

[Andreu et al., 1996] Andreu, Rafael, Ricart, Joan y Valor Josep. "*Estrategia y Sistemas de Información*". 2ª Edition. Serie de Management. McGraw-Hill, 1996.

[Alfonseca et al., 1990] Alfonseca, Manuel, Sancho, Justo y Martínez Orga, Miguel. "*Teoría de Lenguajes, Gramáticas y Autómatas*". Ediciones Universidad y Cultura, 1990.

[Berard, 1996] Berard, Edward V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc. 1996.

[Bertino and Martino, 1993] Bertino, E. and Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

[Boehm, 1989] Boehm, Barry. "*Tutorial on Software Risk Management*". IEEE Computer Society Press, 1989.

[Boehm, 1996] Boehm, Ray. "*Function Point FAQ*". Software Composition Technologies, Inc. June 30, 1996.

[Booch, 1994] Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[CERN, 1997] CERN. "*STING Software Engineering Glossary*". CERN. http://dxsting.cern.ch/sting/glossary.html. April 1997.

[DoD, 1992] DOD. "*DoD Software Reuse Initiative Vision and Strategy*". DOD, 1992.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Freeman, 1987] Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman), IEEE Computer Society Press: 2-8. 1987.

[García y Maudes, 1996] García Peñalvo, Francisco José y Maudes Raedo, Jesús Manuel. "*Introducción a los Algoritmos Genéticos*". ALI Base, Nº28, pp. 49-52. Abril, 1996.

[Ghezzi et al., 1991] Ghezzi, Carlo, Jazayeri, Mehdi and Mandrioli, Dino. "*Fundamentals of Software Engineering*". Prentice-Hall International Inc., 1991.

[Graham, 1994] Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

[Hamilton, 1997] Hamilton, Graham. "*Java Beans Version 1.01*". Sun Microsystems. July, 1997.

[Horan, 1995] Horan, Peter "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Joyanes, 1998] Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Krueger, 1992] Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

[Kuhn, 1971] Kuhn, Thomas S. "*La Estructura de las Revoluciones Científicas*". Fondo de Cultura Económica, 1971.

[Lucey, 1991] Lucey, Terry. "*Management Information Systems*". DP Publications, 1991.

[Madany et al., 1991] Madany, Peter W., Islam, Nayeem, Kougiouris, Panos and Campbell, Roy H. "*Reification and Reflection in C++: An Operating Systems Perspective*". Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA, 1991.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[Martin, 1992] Martin, Robert C. "*Abstract Classes and Pure Virtual Functions*". C++ Report. June/July, 1992.

[Metamodel, 1997] "*Metamodelling Glossary*". http://www.metamodel.com/glossary.html. 1997.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Monforte, 1995] Monforte, Manfredo. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

[Myers, 1978] Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978.

[NIST, 1994] National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

[Peña, 1998] Peña Marí, Ricardo. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Edición. Prentice-Hall, 1998.

[Piattini, 1996] Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

[Piattini et al., 1996] Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pressman, 1992] Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Sarle, 1996] Sarle, Warren S. "*Neural Network FAQ*". June, 1996.

[Schmerl, 1996] Schmerl, Bradley. "*Configuration Management and Version Control*". http://tuvalu.csflinders.edu.au/seweb/scm/lectures/cmvc1.html. 23 May 1996.

[Shaw and Garlan, 1995] Shaw, M., and Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

[Szyperski and Pfister, 1996] Szyperski, Clements and Pfister, Cuno."*First International Workshop on Component-Oriented Programming WCOP'96*". 8 July 1996.

[Tilley et al., 1995] Tilley, Scott, Smith, R. and Dennis B. "*Perspectives on Legacy System Reengineering*". Software Engineering Institute. Draft – Version 0.3. Carnegie Mellon University, Pittsburgh. 1995.

[Web, 1997] Principia Cybernetica Web. "*Web Dictionary of Cybernetics and Systems*". http://pespmc1.vub.ac.be/ASC. 1997.

[Yourdon, 1989] Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Yourdon and Constantine, 1979] Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

Bibliografía

"*Ada 95 Reference Manual: Language and Standard Library*". http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-toc.html [Última vez visitado 8/1/2000].

"*Metamodelling Glossary*". http://www.metamodel.com/glossary.html. 1997.

"*The Pedagogical Patterns Project. Successes in Teaching Object-Technology (PROTO-PATTERNS)*". http://www-lifia.info.unlp.edu.ar/ppp/index.html. [Última vez visitado, 20/8/1999]. July, 1999.

Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.5. ACM/IEEE-CS, October, 1999.

Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.6. ACM/IEEE-CS, February, 2000. Available on line at http://www.swebok.org [Última vez visitado 21-3-2000].

ACM Curriculum Committee on Computer Science. "*Curriculum '68: Recommendations for Academic Programs in Computer Science*". Communications of the ACM, 11(3):151-197. March, 1968.

ACM Curriculum Committee on Computer Curricula of ACM Education Board. "*ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities*". ACM, New York, 1981.

ACM/IEEE-CS. "*Accreditation Criteria for Software Engineering*". http://www.acm.org/serving/se/Accred.htm. [Última vez visitado 28/12/1999]. September, 1998.

ACM/IEEE-CS. "*1999 Plan for the Software Engineering Education Project (SWEEP)*". Draft 0.5. http://www.acm.org/serving/se/sweep.htm. [Última vez visitado 28/12/1999]. April, 1999.

ACM/IEEE-CS. "*Software Engineering Code of Ethics and Professional Practice*". Version 5.2. http://computer.org/tab/sweec/code.htm. [Última vez visitado 28/12/1999]. 1999.

ACM/IEEE-CS. "*Software Engineering Education Project*". http://www.acm.org/serving/se/SWEE.htm. [Última vez visitado 28/12/1999]. 1999.

Adams, Joel C. "*Object-Centered Design. A Five-Phase Introduction to Object-Oriented Programming in CS1-2*". In Proceedings of the twenty-seventh SIGCSE technical

symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 78-82. ACM. 1996.

AENOR. *"Normas para la Gestión y el Aseguramiento de la Calidad*". AENOR, 1992.

ANSI X3J16 and ISO WG21. "*Working Paper for Draft Proposed International Standard for Information Systems—Programming Language C++*". ftp://research.att.com/dist/c++std/WP/CD2. [Última vez visitado 7/1/2000]. X3J16/96–0225 (WG21/N1043). December, 1996.

Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

Aldis, Margaret. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

Alexander, Christopher. "*Notes on the Synthesis of Form*". Harvard University Press, 1964.

Alexander, Christopher. "*The Timeless Way of Building*". The Oxford University Press, 1979.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M. and Angel, S. "*The Oregon Experiment*". Oxford University Press, 1975.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. "*A Pattern Language*". Oxford University Press, 1977.

Alfonseca, Manuel, Sancho, Justo y Martínez Orga, Miguel. "*Teoría de Lenguajes, Gramáticas y Autómatas*". Ediciones Universidad y Cultura, 1990.

Alhir, Sinan Si. "*The Object-Oriented Paradigm*". http://home.earthlink.net/~salhir/theobjectorientedparadigm.html. [Última vez visitado 23/12/1999]. October, 1998.

Ancona, D., Astesiano, E. and Zucca, E. "*Towards a Classification of Inheritance Relations*". Technical Report, Dipartimento di Informatica e Scienze dell'Informazione, Genova. 1992.

Andreu, Rafael, Ricart, Joan y Valor Josep. "*Estrategia y Sistemas de Información*". 2ª Edition. Serie de Management. McGraw-Hill, 1996.

Apple Computer Inc. "*Macintosh Programmer's Workshop Pascal 3.0 Reference*". Apple Computer, 1989.

Appleton, Brad. "*Patterns and Software: Essential Concepts and Terminology*". http://www.enteract.com/~bradapp/docs/patterns-intro.html [Última vez visitado, 15-3-2000]. February, 2000.

Ardis, Mark and Ford, Gary. "*1989 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-89-TR-21 (ESD-TR-89-29), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). June, 1989.

Arnold, Ken and Gosling, James. "*The Java Programming Language*". Addison-Wesley, 1996.

Arnold, Ken and Gosling, James. "*The Java Programming Language*". 2nd edition. Addison-Wesley, 1997.

Ashenhurst, R. L. (Editor). "*Curriculum Recommendations for Graduate Professional Programs in Information Systems*". ACM, 1972.

Ashworth, C. and Goodland, M. "*SSADM: A Practical Approach*". McGraw-Hill, 1990.

Astrachan, Owen, Berry, Geoffrey, Cox, Landon and Mitchener, Garret. "*Design Patterns: An Essential Component of CS Curricula*". In Proceedings of the twenty-ninth SIGCSE

technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 153-160. 1998.

Atkinson, M. and Buneman, P. "*Types and Persistence in Database Programming Languages*". ACM Computing Surveys, 19(2). 1987.

Atkinson, Malcom, Bancilhon, François, DeWitt, David, Dittrich, Klaus, Maier, David, Zdonik, Stanley. "*The Object-Oriented Database System Manifesto*". In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto (Japan). 1989. Also in Deductive and Object-Oriented Databases, Elsevere Science Publishers, Amsterdam, Netherlands, 1990.

Austing, Richard, Barnes, Bruce, Bonnette, Della, Engel, Gerald and Stokes, Gordon. "*Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*". Communications of the ACM, 22(3):147-166. March, 1979.

Bagert, Donald J. "*Talking the Lead in Licensing Software Engineers*". Communications of the ACM, 42(4): 27-29. April, 1999.

Bagert, Donald J., Hilburn, Thomas B., Hislop, Greg, Lutz, Michael, McCracken, Michael and Mengel, Susan. "*Guidelines for Software Engineering Education. Version 1.0*". Working Group on Software Engineering Education and Training (WGSEET). August, 1999.

Bailin, Sidney C. "*An Object-Oriented Requirements Specification Method*". Communications of the ACM, 32(5):608-623. Mayo, 1989.

Balbin, Isaac. "*Is Your Degree Quality Endorsed?*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 60-63. ACM. 1999.

Barr, A. and Feigenbaum, E. "*The Handbook of Artificial Intelligence*". Vol. 1. William Kaufmann, 1981.

Basili, V. R. "*The Future Engineering of Software: A Management Perspective*". IEEE Computer, 24(9):90-96. September, 1991.

Bauer, F. L. "*Software Engineering*". Information Processing 71. Amsterdam: North Holland, 1972.

Beard, R. "*Pedagogía y Didáctica en la Enseñanza Universitaria*". Oikos- Tau, 1974.

Beck, Kent and Cunningham, Ward. "*Using a Pattern Language for Programming*". In Addendum to the Proceedings of OOPSLA'87. ACM SIGPLAN Notices, 23(5). May, 1988.

Beck, Kent and Cunningham, Ward. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in Sigplan Notices, 24(10):1-6. 1989.

Beckman, Kathy. "*Directory of Industry and University Collaborations with a Focus on Software Engineering Education and Training, Version 1*". Special Report CMU/SEI-99-SR-001, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). February, 1999.

Beckman, Kathy, Khajenoori, Soheil, Coulter, Neal and Mead, Nancy R. "*Collaborations: Closing the Industry-Academia Gap*". IEEE Software, 14(6):49-57. November/December, 1997.

Beckman, Kathy, Lawrence, Jimmy, Mead, Nancy, O'Mary, George, Parish, Cynthia and Walker, Hope. "*Industry/University Collaborations: Different Perspectives Heighten Mutual Opportunities*". Software Engineering Institute. http://www.sei.cmu.edu/topics/collaborating/ed/indust-univ-collabs.html. [Última vez visitado, 18/10/1999]. August, 1999.

Bellin, D. "*A Seminar Course in Object-Oriented Programming*". SIGCSE Bulletin, 24(1):134-137. 1992.

Bellin, David. "*Pedagogical Pattern #4. Brainstorming Pattern*". Version 2.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp4.htm. [Última vez visitado, 20/8/1999]. July, 1999.

Berard, Edward V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc. 1996.

Berdugo Gómez de la Torre, Ignacio. "*Candidatura a Rector Encabezada por Ignacio Berdugo Gómez de la Torre. Programa Electoral*". Universidad de Salamanca, 1998.

Bergin, Joseph. "*Pedagogical Patterns*". http://csis.pace.edu/~bergin/PedPat1.2.html. [Última vez visitado, 3/8/1999]. October, 1998.

Bergin, Joseph. "*Pedagogical Pattern #32. Spiral Pattern*". Versión 1.2. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp32.htm. [Última vez visitado, 20/8/1999]. October, 1998.

Bergin, Joseph. "*Six Pedagogical Patterns*". http://csis.pace.edu/~bergin/fivepedpat.html. [Última vez visitado, 3/8/1999]. October, 1998.

Bertino, E. and Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

Bertolino, A. "*KA Description of Software Testing V. 0.5*". In [Abran et al., 1999], 1999.

Bézivin, Jean, Roux, Olivier and Royer, Jean-Claude. "*Teaching Object-Oriented Programming or Using the Object Model to Teach Software Engineering*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 269-275. ACM. 1992.

Biggerstaff, T. J. "*An Assessment and Analysis of Software Reuse*". Advances in Computers. Vol. 34:1-57. 1992.

Birtwistle, Graham M., Dahl, Ole-Johan, Myhrhaug, Bjorn and Nygaard, Kristen. "*Simula Begin*". Studentliteratur, 1973.

Blaha, Michael and Premerlani, William. "*Object-Oriented Modeling and Design for Database Applications*". Prentice Hall, 1998.

Blaha, Michael, Premerlani, William and Rumbaugh, J. "*Relational Database Design Using an Object-Oriented Methodology*". Communications of the ACM, 31(4):414-427. April, 1988.

Blair, G., Gallagher, J., Hutchinson, D. and Shepard, D. "*Object-Oriented Languages, Systems and Applications*". Halsted Press, 1991.

Bloom, B. "*Taxonomy of Educational Objectives: Handbook I: Cognitive Domain*". David McKay, 1956.

Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, 1992.

Bobrow, Daniel G. and Stefik, Mark J. "*LOOPS: an Object-Oriented Programming System for Interlisp*". Xerox PARC, 1982.

Boehm, B. W. "*Software Engineering*". IEEE Transactions on Computers. C-25(12). December, 1976.

Boehm, Barry W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May, 1988.

Boehm, Barry. "*Tutorial on Software Risk Management*". IEEE Computer Society Press, 1989.

Boehm, Ray. "*Function Point FAQ*". Software Composition Technologies, Inc. June 30, 1996.

Bollinguer, Terry. "*Software Construction (Versión 0.5)*". In [Abran et al., 1999], 1999.

Booch, Grady. "*Object Oriented Design with Applications*". The Benjamin/Cummings Publishing Company, 1991.

Booch, Grady. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

Booch, Grady and Rumbaugh, James. "*Unified Method for Object-Oriented Development*". Documentation Set, version 0.8. Rational Software Corporation, 1995.

Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.91 Addendum UML update. Rational Software Corporation, September 1996.

Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997.

Booch, Grady, Jacobson, Ivar and Rumbaugh, James. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0.1. Rational Software Corporation, 19 March 1997.

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

Botella, Pere, Hernández, Juan y Saltor, Fèlix (editores). "*Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*". Grupo de paralelismo del Departamento de Informática de la Universidad de Extremadura. Cáceres, 24-26 de noviembre de 1999.

Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W., Tripp, Leonard, Shyne, Karen, Pflug, Bryan, Maya, Marcela and Tremblay, Guy. "*Guide to the Software Engineering Body of Knowledge. A Straw Man Version*". ACM/IEEE-CS, September, 1998.

Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W. and Tripp, Leonard. "*The Guide to the Software Engineering Body of Knowledge*". IEEE Software, 16(6):35-44. November-December, 1999.

Bourque, Pierre, Dupuis, Robert, Abran, Alain, Moore, James W., Tripp, Leonard and Frailey, Dennis. "*Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge*". Technical Report. January, 1999.

Bowyer, Kevin W. "*Ethics and Computing: Living Responsibly in a Computerized World*". IEEE Computer Society Press, 1996.

Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January, 1990.

Brodsky, S. "*XMI Opens Application Interchange*". White Paper. IBM. March, 1999.

Brown, William H., Malveau, Raphael C., McCormick III, Hays W. and Mowbray, Thomas J. "*Antipatterns. Refactoring Software, Architectures and Projects in Crisis*". Wiley & Sons, 1998.

Bruegge, Bernd and Dutoit, Allen H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Budd, Timothy. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

Budgen, David. "*Software Design Methods: Life Belt or Leg Iron*". IEEE Software, 16(5):133-136. September/October, 1999.

Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad, Peter and Stal, Michael. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Buxton, J. N. and Randell, B. (Editors). "*Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 October, 1969*". Brussels: Scientific Affairs Division, NATO. April, 1970.

Buxton, J. M., Naur, P. and Randell, B. (Editors) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

Cameron, J. "*JSP & JSD: The Jackson Approach to Software Development*". 2nd edition. IEEE Computer Society Press, 1989.

Camps Paré, Rafael. "*¿Qué Informática Se Enseña en la Universidad? (Primera Parte)*". Novática. Nº 141: 48-51. Septiembre/Octubre, 1999.

Cannon, H. I. "*Flavors*". Technical Report, MIT Artificial Intelligence Laboratory, Cambridge (Mass.), 1980.

Carbone, Angela and Kaasbøll, Jens J. "*A Survey of Methods Used to Evaluate Computer Science Teaching*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 41-45. ACM. 1998.

Cardelli, L. and Wegner, P. "*On Understanding Types, Data Abstraction and Polymorphism*". Computing Surveys, 17(4):471-523. December, 1985.

Cardelli, L., Donahue, J., Glassman, L., Jordan, M., Kalsow, B. and Nelson, G. "*Modula-3 Report (revised)*". Technical Report 52, Systems Research Center, Digital Equipment Corporation, Palo Alto. 1989.

Carrington, David. "*SWEBOK Knowledge Area Description for Software Engineering Infraestructure (version 0.5)*". In [Abran et al., 1999], 1999.

Carsí, J. A. "*OASIS como Marco Conceptual para la Evolución de Software*". Tesis Doctoral. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. 1999.

Carter, Janet. "*Collaboration or Plagiarism: GAT Happens when Students Work Together*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in Computer Science Education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 52-55. ACM. 1999.

Castellanos, M. G., Saltor, F. and García-Solaco, M. "*The Development of Semantic Concepts in BLOOM Model Using an Object Metamodel*". Technical Report LSI-91-22. Dept. de Llenguatges i Sistemes Informátics. Universidad Politécnica de Cataluña, 1991.

CERN. "*STING Software Engineering Glossary*". CERN. http://dxsting.cern.ch/sting/glossary.html. April 1997.

Clancy, Michael J. and Linn, Marcia C. "*Patterns and Pedagogy*". In Proceedings of the thirtieth SIGCSE technical symposium on Computer science education, SIGCSE '99. (March 24-28, 1999, New Orleans, LA - USA). Pages 37-42. ACM. 1999.

Coleman, D., Arnold, P., Bodoff, S., Dolin, C., Hayes, F. and Jeremaes, P. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

Concepcion, Arturo I. "*Using an Object-Oriented Software Life-Cycle Model in the Software Engineering Course*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 30-34. 1998.

Consejo de Universidades. "*Plan Nacional de la Calidad de las Universidades. Guía de Evaluación*". Secretaría General del Consejo de Universidades, 1997.

Cook, S. and Daniels, J. "*Designing Object Systems: Object Oriented Modelling with Syntropy*". Prentice-Hall, 1994.

Cook, Steve, Selic, Bran, Gangopadhyay, Dipayan, Gheorge, Serban, Gullekson, Garth, Hogg, John, McGee, Jim, Meier, Mike, Mitra, Subrata, Saaltink, Mark, Warmer Jos and Wills Alan. "*OMG OA&D RFP OMG OA&D RFP Response*". Document Version 1.0. IBM Corporation and ObjecTime Limited. 10 January 1997.

Coplien, James O. "*Advanced C++ Programming Styles and Idioms*", Addison-Wesley, 1992.

Coplien, James O. "*A Pattern Definition - Software Patterns*". http://hillside.net/patterns/definition.html. 1998.

Couger, J. (Editor) "*Curriculum Recommendations for Undergraduate Programs in Information Systems*". Communications of the ACM, 16(12): 727-749. December, 1973.

Cowling, A. J. "*A Multi-Dimensional Model of the Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education and Training – CSEE&T'98. (February 22-25, 1998, Atlanta, GA, USA). Pages 44-55. IEEE Computer Society. 1998.

Cox, Brad J. "*Message/Object Programming: An Evolutionary Change in Programming Technology*". IEEE Software, 1(1):50-69. January, 1984.

Cox, Brad J. and Novobilski, Andrew J. "*Object-Oriented Programming: An Evolutionary Approach*". 2nd edition. Addison-Wesley, 1990.

Crawford, Kathryn and Fekete, Alan. "*What Exams Results Really Measure?*". In Proceedings of the second Australasian conference on Computer science education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. Pages 185-190. 1997.

Crespo González-Carvajal, Yania. "*Incremento del Potencial de Reutilización del Software mediante una Refactorización para Parametrizar*". Tesis Doctoral. Universidad de Valladolid. 2000.

Cybulski, Jacob L. and Linden, Tanya. "*Teaching Systems Analysis and Design Using Multimedia and Patterns*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 113-122. IEEE Press, 2000.

Champeaux, Dennis, Lea, Doug and Faure, Penelope. "*Object-Oriented System Development*". Addison Wesley. 1993.

Chang, Carl K., Engel, Gerald, King, Willis, Roberts, Eric, Shackelford, Russ, Sloan, Robert H. and Srimani, Pradip K. "*Curricula 2001: Bringing the Future to the Classroom*". Computer, 32(9):85-88. September, 1999.

Chen, Peter. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March, 1976.

D'Souza, Desmond F. and Wills, Alan Cameron. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

D'Souza, Desmond F. "*Objects: Education vs. Training*". ICON Computing Inc. http://www.iconcomp.com/papers/education-vs-training/EducationvsTraining.frm.html. [Última vez visitado, 24/5/1999]. 1996.

Dahl, Ole-Johan and Hoare, C. A. R. "*Hierarchical Program Structures*". In Dahl, Dijkstra, Hoare, *Structured Programming*, Academic Press, pages 175-220. 1972.

Dahl, Ole-Johan and Nygaard, Kristen. "*SIMULA — An Algol-Based Simulation Language*". Communication of the ACM, 9(9):671-678. September, 1966.

Dahl, Ole-Johan, Myrhaug, Bjorn and Nygaard, Kristen. "*(Simula 67) Common Base Language*". Norsk Regnesentral (Norwegian Computing Center), Publication N. S-22, Oslo. October, 1970.

Danforth, S. and Tomlinson, C. "*Type Theories and Object-Oriented Programming*". ACM Computing Surveys, 20(1):29-72, 1988.

Data Processing Management Association. "*DPMA Model Curriculum, 1981*". Published by DPMA, Chicago, 1981.

Davis, Alan M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

Davis, Gordon B., Gorgone, John T., Couger, J. Daniel, Feinstein, David L. and Longenecker, Jr. Herbert E. (editors) "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS and AITP, 1997.

Davis, William S. "*Tools and Techniques*". Addison-Wesley, 1983.

Dawson, Ray and Newsham, Ron. "*Introducing Software Engineers to the Real World*". IEEE Software, 14(6):37-43. November/December, 1997.

Decker, Rick and Hirshfield, S. "*The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS1*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 51-55. ACM. 1994.

DeClue, Tim. "*Object-Oriented and the Principles of Learning Theory: A New Look at Problems and Benefits*". In Proceedings of the twenty-seventh SIGCSE technical

symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 232-236. ACM. 1996.

DeMarco, Tom "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

Denning, Peter J. "*Educating a New Engineer*". Communications of the ACM, 35(12). December, 1992.

Denning, Peter J. "*Computer Sciencie and Software Engineering: Filing for Divorce?*". Communications of the ACM, 40(8):128. August, 1998.

Denning, Peter J., Comer, Douglas E., Gries, David, Mulder, Michael C., Tucker, Allen B., Turner, A. Joe and Young, Paul R. "*Report of the ACM Task Force on the Core of Computer Science*". ACM Press, 1988.

Denning, Peter J., Comer, Douglas E., Gries, David, Mulder, Michael C., Tucker, Allen B., Turner, A. Joe and Young, Paul R. "*Computing as a Discipline*". Communications of the ACM, 32(1):9-23. January, 1989.

Deveaux, Daniel, Fleurquin, Regis and Frison, Patrice. "*Software Engineering Teaching: A 'Docware' Approach*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 163-166. ACM. 1999.

Díaz, Oscar. "*¿Para Qué la Tesis Doctoral?*". Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISDB'99). (Cáceres, 24-25 de noviembre de 1999). Páginas 3-6. 1999.

Díaz, Oscar y Lopistéguy, Philippe (editores). "*Actas de las II Jornadas de Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad del País Vasco/Euskal Herriko Unibertsitatea. Donosita-San Sebastián, 2-5 de septiembre de 1997.

Díaz-Herrera, Jorge and Powell, Gerald M. "*Educating Industrial-Strength Software Engineers*". In Proceedings of the 11th Conference on Software Engineering Education and Training (CSEE&T '98). (February 22-25, 1998. Atlanta, GA – USA). IEEE Computer Society. Pages 139-150. 1998.

Dijkstra, E. "*The Structure of 'THE' Multiprogramming System*". Communications of the ACM, 11(5). May, 1968.

Diller, A. "*Z: An Introduction to Formal Methods*". John Wiley & Sons, 1990.

DOD. "*DoD Software Reuse Initiative Vision and Strategy*". DOD, 1992.

Donadi, Mahesh H. "*Teaching Practical Object-Oriented Software Engineering*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 251-256. ACM. 1992.

Dodani, Mahesh. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January, 1999.

Dolado, Javier. "*El Código de Ética y Práctica Profesional de la Ingeniería del Software de la ACM/IEEE Computer Society*". Novática. Nº 140. Julio-Agosto, 1999.

Dorling, Alec. "*Software Process Improvement and Capability Determination*". Software Quality Journal, 12(4): 209-224. December, 1993.

Duncan, William R. "*A Guide to the Project Management Body of Knowledge*". PMI Standards Committee. Project Management Institute, Four Campus Boulevart, Newtown Square, PA 19073-3299, USA. 1996.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para el Análisis de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/norma/analisis-2.zip. [Última vez visitado, 10-3-2000]. Sevilla, 29 de noviembre de 1999.

Durán Toro, Amador y Bernárdez Jiménez, Beatriz. "*Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.0*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/norma/elicitacion.zip. [Última vez visitado, 10-3-2000]. Sevilla, 18 de octubre de 1999.

Education Activities Board. "*The 1983 Model Program in Computer Science and Engineering*". Technical Report 932. IEEE Computer Society. December, 1983.

Ehrig, H. and Mahr, B. "*Fundamentals of Algebraic Specification 1*". Springer-Verlag, EATCS Nº6, 1985.

EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

El-Kadi, Amr. "*Stop that Divorce*". Communications of the ACM, 42(12):27-28. December, 1999.

Ellis, Margaret and Stroustrup, Bjarne. "*The Annotated C++ Reference Manual*". Addison Wesley, 1990.

Fairley, Richard. "*Software Engineering Concepts*". McGraw-Hill, 1985.

Fell, Harriet J., Proulx, Viera K. and Casey, John. "*Writing across the Computer Science Curriculum*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 204-209. ACM. 1996.

Fell, Harriet J., Proulx, Viera K. and Rasala, Richard. "*Scaling: A Design Pattern in Introductory Computer Science Courses*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 326-330. 1998.

Firesmith, Donald, Henderson-Sellers, Brian and Graham, Ian. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

Fisher, Alan S. "*CASE: Using Software Development Tools*". 2nd Edition. John Wiley & Sons, 1991.

Ford, Gary. "*1990 SEI Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-90-TR-3 (ESD-TR-90-204), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March, 1990.

Ford, Gary. "*1991 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-91-TR-2 (ESD-TR-91-2), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April, 1991.

Ford, Gary. "*The SEI Undergraduate Curriculum in Software Engineering*". In Proceedings of the twenty-second SIGCSE technical symposium on Computer Science Education – SIGCSE'91. (March 7-8, 1991, San Antonio, Texas, USA). Pages 375-385. ACM. 1991.

Ford, Gary. "*A Progress Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-94-TR-11 (ESC-TR-94-011), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). May, 1994.

Fowler, Martin. "Analysis Patterns: Reusable Object Models". Object Technology Series. Addison-Wesley, 1996.

Fowler, Martin and Scott, Kendall. "*UML Distilled. Applying the Standard Object Modeling Language*". Object Technology Series. Addison Wesley, 1997.

Fowler, Martin and Scott, Kendall. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd Edition. Object Technology Series. Addison Wesley, 2000.

Frakes, William B., Fox, Christopher, Nejmeh and Brian A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman), IEEE Computer Society Press: 2-8. 1987.

Gamma, Erich. "*Objektorientierte Software-Entwicklung am Beispiel von ET++: Klassenbibliothek, Werkzeuge, Desing*". Dissertation, Universität Zürich, 1991.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns: Abstraction and Reuse of Object-Oriented Design*". In Proceedings of the 7th European Conference in Object Oriented Programming – ECOOP'93. Nierstrasz, O. M. editor. Pages 406-431. Springer Verlag, 1993.

Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

Gane, C. and Sarson, T. "*Structured Systems Analysis and Design*". Improved Systems Technologies, Inc., 1977.

Gane, C. and Sarson, T. "*Structured Systems Analysis: Tools and Techniques*". Prentice-Hall, 1979.

Gane, Chris and Sarson, Trish. "*Análisis Estructurado de Sistemas*". Ateneo, 1981.

Garbajosa, Juan y Bonilla, A. "*Integración de Herramientas CASE*". Capítulo 22 en [Piattini y Daryanani, 1995]. 1995.

García Carrasco, Joaquín, García del Dujo, Ángel, López Fernández, Ricardo, Mompó Gómez, Rafael, Navazo Suela, María Agustina, Pérez Juárez, María Ángeles, Redoli Granados, Judith, Regueras Santos, Luisa María, Rodríguez Pajares, Blanca y Verdú Pérez, María Jesús. "*Nuevas Tecnologías y Formación*". PCWEEK. Editorial América Ibérica, Madrid. 1999.

García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1996-1997. 1996.

García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Programación Avanzada*". Tercer Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

García Peñalvo, Francisco José. "*Patrones. De Alexander a la Tecnología de Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(10):44-52. Noviembre, 1998. (También disponible en http://tejo.usal.es/~fgarcia/doc/patrones1.pdf).

García Peñalvo, Francisco José. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

García Peñalvo, Francisco José. "*Plan de Calidad para la Asignatura Ingeniería del Software*". Tercer Curso de la Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Curso 1998-1999. 1999.

García Peñalvo, Francisco José. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

García Peñalvo, Francisco José y Maudes Raedo, Jesús Manuel. "*Introducción a los Algoritmos Genéticos*". ALI Base, Nº28, pp. 49-52. Abril, 1996.

García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report TR-GIRO-01-97V2.1.1. Versión 2.1.1. Universidad de Valladolid. 1997.

García Peñalvo, Francisco José, Montero García, Eduardo y Arranz Val, Pablo. "*Proceso de Evaluación por Pares. Una Experiencia Práctica*". En las actas de las II Jornadas de Calidad y Universidad: Calidad en la Docencia (Burgos, 10-11 de Noviembre de 1998).

García Peñalvo, Francisco José, Moreno García, María N., González Talaván, Guillermo y Moreno Montero, Ángeles María. "*Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 46 y ponencia en versión digital (CD-ROM). 1999.

García Peñalvo, Francisco José, Moreno García, María N., Montero García, Eduardo y Arranz Val, Pablo. "*Evaluación del Profesorado: Un Protocolo de Evaluación por Pares*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 47 y ponencia en versión digital (CD-ROM). 1999.

García Peñalvo, Francisco José, Maudes Raedo, Jesús Manuel, Piattini Velthuis, Mario Gerardo, García-Bermejo Giner, José Rafael y Moreno García, María N. "*Proyecto de Final de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación*". Departamento de Informática y Automática de la Universidad de Salamanca. Versión 1.52. http://tejo.usal.es/~fgarcia/doc/pfc.pdf. Marzo, 2000.

García Peñalvo, Francisco José, Moreno García, María N., García-Bermejo Giner, José Rafael y Luis Reboredo, Ana de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

Garlan, David, Gluch, David P. and Tomayko, James E. "*Agents of Change: Educating Software Engineering Leaders*". IEEE Computer, 30(11):59-65 November, 1997.

Gelfand, Natasha, Goodrich, Michael T. and Tamassia, Roberto. "*Teaching Data Structure Design Patterns*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 331-335. 1998.

Gersting, Judith L. "*A Software Engineering 'Frosting' on a Traditional CS-1 Course*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 233-237. ACM. 1994.

Ghezzi, C. and Jazayeri, M. "*Programming Languages Concepts*". 3rd Edition. John Wiley & Sons, 1998.

Ghezzi, Carlo, Jazayeri, Mehdi and Mandrioli, Dino. "*Fundamentals of Software Engineering*". Prentice-Hall International Inc., 1991.

Gibbs, Norman E. "*The SEI Education Program: The Challenge of Teaching Future Software Engineers*". Communications of the ACM, 32(5):594-605. May, 1989.

Gibbs, N. E. and Tucker, A. B. "*Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 29(3):202-210. March, 1986.

Glass, Robert L. "*The Relationship between Theory and Practice in Software Engineering*". Communications of the ACM, 39(11):11-13. November, 1996.

Glass, Robert L. "*Revisiting the Industry/Academe Communication Chasm*". Communications of the ACM, 40(6): 11-13. June, 1997.

Goguen, J. "*Parametrized Programming*". IEEE Transactions on Software Engineering, 10(5):528-543, 1984.

Goguen, J. A. and Meseguer, J. "*Order-Sorted Álgebra I*". Technical Report, SRI International, Stanford University, 1988.

Goguen, J. A., Winkler, T., Meseguer, J., Futatsugui, K. and Jouannaud, J. P. "*Introducing OBJ*". SRI-CSL Report. Draft of January, 1992.

Goldberg, Adele. "*Smalltalk-80: The Interactive Programming Environment*". Addison-Wesley, 1985.

Goldberg, Adele and Kay, Alan. "*Smalltalk-72 Instruction Manual*". Technical Report SSL-76-6, Xerox Palo Alto Research Center. March, 1976.

Goldberg, Adele and Robson, David. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

Goldberg, R. "*Software Engineering: An Emerging Discipline*". IBM Systems Journal. 25(3/4), 1986.

Gómez, A., Juristo, N., Montes, C. y Pazos, J. "*Ingeniería del Conocimiento*". Madrid. Ceura, 1998.

Gómez Pérez, R. Prólogo de la obra de J. Pujol y J. P. Fons. "*Los Métodos de Enseñanza Universitaria*". Eunsa. 1981.

González Casal, J. "*Estudio de la Profesión Informática en España durante 1995*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (29):13-16. 1996.

Gorgone, John T., Gray, Paul, Feinstein, David L., Kasper, George M., Luftman, Jerry N., Stohr, Edward A., Valacich, Joseph and Wigand, Rolf T. "*MSIS 2000 Model*

Curriculum and Guidelines for Graduate Degree Programs in Information Systems". ACM and AIS. November, 1999.

Gotterbarn, D. "*How the New Software Engineering Code of Ethics Affects You*". IEEE Software, 16(6):58-64. November/December, 1999.

Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics*". Communications of the ACM, 40(11):110-118. November, 1997.

Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics, Version 3.0*". IEEE Computer, 30(11):88-92. November, 1997.

Gotterbarn, D., Miller, K. and Rogerson, S. "*Computer Society and ACM Approve Software Engineering Code of Ethics*". IEEE Computer, 32(10):84-88. October, 1999.

Gotterbarn, D., Miller, K. and Rogerson, S. "*Software Engineering Code of Ethics Is Approved*". Communications of the ACM, 42(10):102-107. October, 1999.

Graham, Ian. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

Graham, Ian M. "*Migrating to Object Technology*". Addison-Wesley, 1995.

Graham, Ian, Bischof, Julia and Henderson-Sellers, Brian. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February, 1997.

Graham, Ian, Henderson-Sellers, Brian and Younessi, Houman. "*The Open Process Specification*". Addison Wesley (Open Series), 1997.

Granger, Mary J. and Little, Joyce Currie. "*Integrating CASE Tools into the CS/CIS Curriculum*". In Proceedings of the conference on Integrating technology into computer science education, ITiCSE '96. (June 2-6, 1996, Barcelona, Spain). Pages 130-132. ACM, 1996.

Greening, Tony. "*Examining Student Learning of Computer Science*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 63-66. ACM. 1997.

Griss, Martin L. "*The Architecture and Processes for Systematic OO Reuse Factory*". In Proceedings of the 7th Workshop on Institutionalizing Software Reuse (WISR-7), Andersen Consulting's Center for Professional Education, St. Charles, Illinois, 28-30 August 1995.

Guerraoui, R. (editor), Aksit, M., Black, A., Cardelli, L., Cointe, P., Coplien, J., Kiczales, G., Lea, D., Madsen, O., Magnusson, B., Meseguer, J., Moessenboeck, H., Palsberg, J. and Schmidt, D. "*Strategic Research Directions in Object Oriented Programming*". Technical Report based on position statements held during ACM Workshop on Strategic Directions in Computing. MIT, June, 1996.

Gullbert, J. J. "*Guía Pedagógica*". Organización Mundial de la Salud. Quinta Edición. Editado por Instituto de Ciencias de la Educación. Universidad de Valladolid. 1989.

Guttag, J. "*Abstract Data Types and the Development of Data Structures*". In *Programming Language Design*. Computer Society Press, 1980.

Hadjerrouit, Said. "*A Constructivist Approach to Object-Oriented Design and Programming*". In Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 171-174. ACM. 1999.

Halbert, D. and O'Brien, P. "*Using Types and Inheritance in Object-Oriented Programs*". IEEE Software, pages 71-79. 1987.

Hale. "*Reports of the Committee on University Teaching Methods"*. Londres, University Grants Committee, 1964.

Hamilton, Graham. "*Java Beans Version 1.01*". Sun Microsystems. July, 1997.

Harbison, Samuel P. "*Modula-3*". Prentice Hall, 1992.

Hatley D. J. and Pirbhai, A. "*Strategies for Real-Time System Specification*". Dorset House, 1987.

Henderson-Sellers, B. "*OPEN Relationships – Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December, 1997.

Henderson-Sellers, B. and Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September, 1990.

Henderson-Sellers, B. and Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

Henderson-Sellers, B. and Edwards, J. M. "*MOSES: A Second Generation Object-Oriented Methodology*". Object Magazine, pp. 68-73. June, 1994.

Henderson-Sellers, Brian, Simons, Anthony, Younessi, Houman. "*The Open Toolbox of Techniques*". Open Series. Addison Wesley, 1998.

Hilburn, Thomas B. "*Software Engineering Education: A Modest Proposal*". IEEE Software, 14(6):44-48. November/December, 1997.

Hilburn, Thomas B., Bagert, Donald J., Mengel, Susan and Oexmann, Dale. "*Software Engineering Across Computing Curricula*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 117-121. ACM. 1998.

Hilburn, Thomas B., Hirmanpour, Iraj, Khajenoori, Soheil, Turner, Richard and Qasem, Abir. "*A Software Engineering Body of Knowledge Version 1.0*". Technical Report CMU/SEI-99-TR-004 (ESC-TR-99-004), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April, 1999.

Hirmanpour, Iraj, Hilburn, Thomas B. and Kornecki, Andrew. "*A Domain Centered Curriculum. An Alternative Approach to Computing Education*". In Proceedings of the 26th SISCSE technical symposium on Computer Science Education, SIGCSE '95. (March 2-4, 1995, Nashville, TN, USA). ACM. 1995.

Hoare, C. A. R. "*Communicating Sequential Processes*". Prentice-Hall, 1985.

Holland, Simon, Griffiths, Robert and Woodman, Mark. "*Avoiding Object Misconceptions*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 131-134. 1997.

Horan, Peter "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

Hullot, Jean-Marie. "*Ceyx, Version 15: I — une Initiation*". Rapport Technique no. 44, INRIA, Rocquencourt, 1984.

Humphrey, W. S. "*Managing the Software Process*". Addison-Wesley, 1989.

Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science,* Van Nostrand Reinhold, p. 1218,1993.

IEEE. "*Standard Glossary of Software Engineering Terminology*". ANSI/IEEE Std. 729-1983. IEEE, 1983.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 4: Resource and Technique Standards*". IEEE Computer Society Press, 1999.

Ingalls, Daniel H. "*The Smalltalk-76 Programming System: Design and Implementation*". In Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages. ACM. January, 1978.

ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

ISO/IEC. "*Programming Languages – C++*". Technical ISO/IEC 14882. September, 1998.

Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

Jackson, M. A. "*System Development*". Prentice-Hall, 1983.

Jackson, Michael. "*Critical Reading for Software Developers*". IEEE Software, 12(6):103-104. November, 1995.

Jackson, Ursula, Manaris, Bill and McCauley, Renée. "*Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum.*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education, SIGCSE '97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 360-364. ACM. 1997.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Jacobson, I., Booch, G. and Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

Jalics, P. and Golden, D. "*A Profile of Undergraduate Computer Science Curricula*". Computer Science Education, 6(2):179-192. November, 1995.

Jalloul, Ghinwa. "*Pedagogical Pattern #48. Academic to Industrial Project Link (LINK) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp48.htm. [Última vez visitado, 20/8/1999]. July, 1999.

Jaworski, Jaime. "*Java 1.2 Unleashed*". Sams, 1998.

Jézéquel, Jean-Marc and Meyer, Bertrand. "*Design by Contract: The Lessons of Ariane*". IEEE Computer, 30(1):129-130. January, 1997.

Jones, A. K. "*The Object Model: A Conceptual Tool for Structuring Software*". In Gerald E. Peterson, editor, *TUTORIAL: Object-Oriented Computing*, volume 2: Implementations. IEEE Computer Society Press, 1987.

Jones, C. B. "*Systematic Software Construction Using VDM*". Prentice-Hall, 1990.

Joyanes Aguilar, Luis. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

Joyner, Ian. "*C++?? A Critique of C++ and Programming and Language Trends of the 1990s*". 3rd edition http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/cppcv3.pdf. [Última vez visitado 7/1/2000]. 1996.

Joyner, Ian. "*Objects Unencapsulated. Java™, Eiffel, and C++??*". Object and Component Technology Series. Prentice Hall, 1999.

Karlsson, Even-André (editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

Kernighan, Brian W. and Ritchie, Dennis M. "*The C Programming Language*". Prentice-Hall, 1988.

Knight, John C., Prey, Jane C. and Wulf, Wm. A. "*Undergraduate Computer Science Education: A New Curriculum Philosophy & Overview*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 155-159. ACM. 1994.

Knudsen, J. L. and Madsen, O. L. "*Teaching Object-Oriented Programming Languages*". In Proceedings of ECOOP'98. Springer-Verlag. Pages 21-40. August, 1988.

Kobryn, Cris. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October, 1999.

Koffman, E., Miller, P. and Wardle, C. "*Recommended Curriculum for CS1: 1984*". Communications of the ACM, 27(10):998-1001. October, 1984.

Koffman, E., Miller, P. and Wardle, C. "*Recommended Curriculum for CS2: 1984*". Communications of the ACM, 28(8):815-818. August, 1985.

Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 1: Languages*". Journal of Object-Oriented Programming, 11(8):8-15. January, 1999.

Kölling, Michael. "*The Problem of Teaching Object-Oriented Programming, Part 2: Environments*". Journal of Object-Oriented Programming, 11(9):6-12. February, 1999.

Konrad, Michael D., Paulk, Mark C., and Graydon, Allan W. "*An Overview of SPICE's Model for Process Management*". In Proceedings of the Fifth International Conference on Software Quality, Austin, TX, 23-26 October 1995.

Krasner, G. E. and Pope, S. T. "*A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*". Journal of Object-Oriented Programming (JOOP), SIGS Publications, 1(3):26-49. August-September, 1988.

Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June, 1992.

Kruglinski, D. "*Inside Visual C++*". Microsoft Press, 1995.

Kuhn, Sarah. "*The Software Design Studio: An Exploration*". IEEE Software, 15(2):65-71. March/April, 1998.

Kuhn, Thomas S. "*La Estructura de las Revoluciones Científicas*". Fondo de Cultura Económica, 1971.

Lafourcade, P. "*Planteamiento, Conducción y Evaluación de la Enseñanza Universitaria*". Kapeluzs, 1974.

Lalonde, Wilf R. and Pugh, John R. "*Inside Smalltalk*". Vol. 1. Prentice-Hall, 1990.

Lalonde, Wilf R. and Pugh, John R. "*Inside Smalltalk*". Vol. 2. Prentice-Hall, 1991.

Lara Ortega, Fernando. "*Principios de Calidad en la Docencia*". Actas de las I Jornadas sobre Calidad y Universidad. Universidad de Burgos. Noviembre, 1997.

Larman, Craig. "*Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*". Prentice Hall, 1998.

Layman, B. "*ISO-9000 Standards and Existing Quality Models: How They Relate*". American Programmer Review, pp. 9-15. February, 1994.

Le Moigne, J. L. "*Les Systemes D'Information dan les Organizations*". Presses Universitaires de France, 1973.

Lea, Doug. "*Christopher Alexander: An Introduction for Object-Oriented Designers*". Software Engineering Notes, ACM SIGSOFT, 19(1):39-46, January 1994 (Online version: http://g.oswego.edu/dl/ca/ca/ca.html).

Lebsanft, E. and Synspace, A. G. "*BOOTSTRAP: Experiences with Europe's Software Process Assesment & Improvement Method*". In Proceedings of the 1st World Congress for Software Quality, 1994.

Letelier, P., Ramos, I., Sánchez, P., y Pastor, O. "*OASIS Versión 3.0. Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-98.4011. 1998.

Leveson, Nancy G. "*Software Engineering: Stretching the Limits of Complexity*". Communications of the ACM 40(2): 129-131. February, 1997.

Levine, Linda, Pesante, Linda H. and Dunkle, Susan B. "*Technical Writing for Software Engineers*". Curriculum Module, SEI-CM-23, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). November, 1991.

Levy, H. "*Capability-Based Computer Systems*". Digital Press, 1984.

Lilly, Susan. "*Patterns for Pedagogy*". Object Magazine, 5(8):93-96. January, 1996.

Lippman, S. "*C++ Primer*". Addison-Wesley, 1989.

Liskov, Barbara. "*Data Abstraction and Hierarchy*". In Addedum to Proceedings of OOPSLA'87. Pages 17-35. ACM Press, 1987.

Liskov, Barbara. "*A History of CLU*". In Proceedings of the second ACM SIGPLAN conference on History of programming languages – HOPL II. (April 20 - 23, 1993, Cambridge United States). ACM SIGPLAN Notices, 28(3):133-147. March, 1993.

Liskov, B. and Zilles, S. "*An Introduction to Formal Specifications of Data Abstractions*". Current Trends in Programming Methodology: Software Specification and Design. Vol. 1. Prentice-Hall, 1977.

Lloyd, D. H. "*A Concept of Improvement of Learning Response in the Taught Lesson*". Visual Education, 1968.

Longenecker, Jr. Herbert E. and Feinstein, David L. (Editors) "*IS'90 The DPMA Model Curriculum for a Four Year Undergraduate Degree for the 1990s*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1991.

Longenecker, Jr. Herbert E., Fournier, Robert, Reaugh, William R. and Feinstein David L. (Editors) "*IS'94 The DPMA Two Year Model Curriculum for IS Professionals*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1994.

Lucey, Terry. "*Management Information Systems*". DP Publications, 1991.

Luker, P. "*There's More to OOP than Syntax!*". SIGCSE Bulletin, 26(1):56-60. 1994.

Lutz, Michael J. and Naveda, J. Fernando. "*The Road Less Traveled: A Baccalaureate Degree in Software Engineering*". In Proceedings of the twenty-eighth SIGCSE Technical

Symposium on Computer Science Education (SIGCSE'97). (February 27 - March 1, 1997, San Jose, CA USA). ACM. Pages 287-291. 1997.

Madany, Peter W., Islam, Nayeem, Kougiouris, Panos and Campbell, Roy H. "*Reification and Reflection in C++: An Operating Systems Perspective*". Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA, 1991.

Madsen, Ole Lehrmann, Møller-Pedersen, Birger, Nygaard, Kristen. "*Object-Oriented Programming in the BETA Programming Language*". Addison-Wesley, Wokingham (U.K.), 1993.

Mager, R. F. "*Creación de Actitudes y Aprendizajes*". 2ª edición. Colección Biblioteca del Educador. Editorial Marova. 1976.

Manns, Mary Lynn. "*Pedagogical Pattern #8. Lab-Discussion-Lecture-Lab (LDLL) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp8.htm. [Última vez visitado, 20/8/1999]. July, 1999.

Manso Martínez, J. M. "*Docencia en la Universidad: Lo que Es y lo que Debe Ser*". Actas de las I Jornadas sobre Calidad y Universidad – Hacia una Universidad de Calidad. Burgos, 10-13 de Nov. de 1997.

Marchionini, Gary and Maurer, Hermann. "*The Roles of Digital Libraries in Teaching and Learning*". Communications of the ACM, 38(4):67-75. April, 1995.

Marqués Corral, José Manuel. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

Marqués Corral, José Manuel. "*Proyecto Docente e Investigador. Ingeniería del Software e Inteligencia Artificial*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Octubre, 1998.

Marqués Corral, José Manuel. "*Estándares de documentación. Laboratorio de Ingeniería del Software I. Ingeniería Técnica en Informática de Sistemas de la Universidad de Valladolid*". Departamento de Informática de la Universidad de Valladolid. Septiembre, 1999.

Martin, James and Odell, James J. "*Object-Oriented Methods: A Foundation*". Prentice Hall, 1995.

Martin, James and Odell, James J. "*Object-Oriented Methods: Pragmatic Considerations*". Prentice Hall, 1996.

Martin, James and Odell, James J. "*Object-Oriented Methods: A Foundation, UML Edition*". 2nd Edition. Prentice Hall, 1998.

Martin, Robert C. "*Abstract Classes and Pure Virtual Functions*". C++ Report. June/July, 1992.

Martin, Robert C. "*The Open Closed Principle*". C++ Report, 8(1). January, 1996.

Martin, Robert C. "*The Liskov Substitution Principle*". C++ Report, 8(3). March, 1996.

Martin, Robert C. "*The Dependency Inversion Principle*". C++ Report, 8(5). May, 1996.

Martin, Robert C. "*The Interface Segregation Principle*". C++ Report, 8(7). July-August, 1996.

Martin, Robert C. "*Granularity*". C++ Report, 8(10). November-December, 1996.

Martin, Robert C. "*Principles of OOD*". OMA. 1997.

Martin, Robert C. "*Stability*". C++ Report, 9(2). February, 1997.

Mason, David V. and Woit, Denise M. "*Integrating Technology into Computer Science Examinations*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 140-144. 1998.

Maudes Raedo, Jesús Manuel. "*Proyecto Docente para las Asignaturas: Sistemas de Gestión de Bases de Datos y Administración de Bases de Datos*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Escuela Universitaria Politécnica. Universidad de Burgos. Noviembre, 1998.

McCauley, Renée A., Jackson, Ursula and Manaris, Bill. "*Documentation Standards in the Undergraduate Computer Science Curriculum*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 242-246. ACM. 1996.

McClure, Carma. "*Experiences from the OO Playing Field*". The Object World Insider, 2(4):1-3. 1996.

McClure, Carma. "*Software Reuse Techniques: Adding Reuse to the System Development Process*". Prentice Hall, 1997.

McDonald, Gary and McDonald, Merry. "*Developing Oral Communication Skill of Computer Science Undergraduates*". In Proceedings of the twenty-fourth SIGCSE technical symposium on Computer Science Education, SIGCSE '93. (Feb. 18-19, 1993, Indianapolis, IN, USA). Pages 279-282. ACM. 1993.

Mead, Nancy, Unpingco, Perla, Beckman, Kathy, Walker, Hope, Parish, Cynthia and O'Mary, George. "*Industry/University Collaborations. Different Perspectives Heighten Mutual Opportunities*". Crosstalk, The Journal of Defense Software Engineering, 13(3):10-15. March, 2000.

Mercuri, Rebecca. "*In Search of Academy Integrity*". Communications of the ACM, 40(5):136. May, 1998.

Meyer, Bertrand. "*Object Oriented Software Construction*". Prentice Hall, 1988.

Meyer, Bertrand. "*Eiffel: The Language*". Object-Oriented Series. Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

Meyer, Bertrand. "*Reusable Software. The Base Object-Oriented Component Libraries*". The Object-Oriented Series. Prentice Hall, 1994.

Meyer, Bertrand. "*Teaching Object Technology*". IEEE Computer, 29(12):117. December, 1996.

Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

Miguel, Adoración de y Piattini, Mario G. "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al ModeloRelacional*". Ra-ma, 1993.

Miller, Jan and Mingins, Christine. "*Putting the Practice into Software Education*". In Proceedings oh the 1998 International Conference on Software Engineering: Education and Practice (SEEP'98). (26-29 January 1998, Dunedin – New Zeland). Pages, 200-208. IEEE Computer Society. 1998.

Ministerio de Educación y Ciencia. "*Ley Orgánica de Reforma Universitaria*". Servicio de Publicaciones del MEC, 1993.

Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

Minsky, M. "*A Framework for Representing Knowledge*". In Haugeland, J. *Mind Design*. MIT Press, 1981.

Mohedano, José Eduardo. "*Java en la Historia: El Azar Mueve el Mundo*". Sólo Programadores. Especial Monográfico Nº3: 6-10. Octubre, 1998.

Moitra, Deependra. "*Software Engineering in the Small. Practical Software Engineering and Management*". IEEE Computer, 32(10):39-40. October, 1999.

Monarchi, David E. and Puhr, Gretchen I. "*A Research Typology for Object-Oriented Analysis and Design*". Communicatios of the ACM, 35(9):35-47. September, 1992.

Monforte, Manfredo. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

Mora Núñez, N., Prima Rodríguez, M., González López, R., Crespo Faja, F. y Díaz Vázquez, J. E. "*Propuesta de Organización Docente para Asignaturas Troncales de Pocos Créditos, Basada en Principios Constructivistas*". Actas de las III Jornadas Universitarias sobre Innovación Educativa en las Enseñanzas Técnicas (Ferrol – A Coruña. Septiembre, 1995). Tomo II. Páginas 285-292. 1995.

Moreno Montero, Ángeles María. "*Proyecto Docente. Redes de Ordenadores y Bases de Datos*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Abril, 2000.

Moreno Rodilla, Vidal. "*Proyecto Docente e Investigador. Control e Instrumentación de Procesos Químicos*". Área de Conocimiento de Ingeniería de Sistemas y Automática. Facultad de Ciencias. Universidad de Salamanca. Junio, 1999.

Musser, D. and Saini, A. "*STL Tutorial and Reference Guide*". Addison-Wesley, 1996.

Musser, D. R. and Stepanov, A. A. "*A Library of Generic Algorithms in Ada*" Proceedings of 1987 ACM SIGAda International Conference, Boston, December, 1987.

Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". In Proceedings of the First International Joint Conference of ISSAC-88 and AAECC-6. P. Gianni editor. (Rome, Italy, July 4-8, 1988). Published in *Lecture Notes in Computer Science* 358, Pages 13-25. Springer-Verlag, 1989.

Musser, David R. and Stepanov, Alexander A. "*The Ada Generic Library: Linear List Processing Packages*". Compass Series. Springer-Verlag, 1989.

Musser, David R. and Stepanov, Alexander A. "*Algorithm-Oriented Generic Library*". Software Practice & Experience, 24(7):623-642. July, 1994.

Musser, David R. and Stepanov, Alexander A. "*Generic Programming*". Dagstuhl Seminar on Generic Programming. 1998.

Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978.

Nachbar, Daniel. "*Bringing Real-World Software Development into the Classroom: A Proposed Role for Public Software in Computer Science Education*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 171-175. 1998.

National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

Naughton, Patrick. "*The Java Handbook*". McGraw-Hill, 1996.

Naur, P. and Randell, B. (Editors). "*Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968*". Brussels: Scientific Affairs Division, NATO. January, 1969.

Nerson, J. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September, 1992.

Neumann, Peter G. "*Risks of E-Education*". Communications of the ACM, 40(10):136. October, 1998.

Nguyen, Dung. "*Design Patterns for Data Structures*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 336-340. 1998.

Nishida, Tomohiro, Saitoh, Akinori, Tsujino, Yoshihiro and Tokura, Nobuki. "*Lecture Supporting System by E-mail and WWW*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 280-284. ACM. 1996.

Norman, D. A. "*The Psychology of Everyday Things*". New York: Basic Books, Inc. 1988.

Northrop, Linda M. "*Finding an Educational Perspective for Object-Oriented Development*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 245-249. ACM. 1992.

Null, Linda. "*Applying TQM in the Computer Science Classroom*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 120-124. ACM. 1996.

Nunamaker, Jay F., Couger, J. D. and Davis, Gordon B. "*Information System Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs*". Communications of the ACM 25(11). November, 1982.

Nygaard, Kristen and Dahl, Ole-Johan. "*The Development of the SIMULA Language*". In *History of Programming Languages*, Richard L. Wexelblat editor. Pages 439-493. Academic Press, 1981.

OMG. "*OMG Unified Modeling Language Specification. Version 1.2*". Object Management Group Inc. ftp://ftp.omg.org/pub/docs/ad/98-12-02-pdf. July, 1998.

OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June, 1999.

Orden Hoz, A. de la. "*Modelos de Evaluación Universitaria*". Revista Española de Pedagogía. Nº 169. 1985.

Orr, K. T. "*Structured System Development*". Yourdon Press, 1977.

Orr, K. T. "*Structured Requirements Definition*". Ken Orr & Associates, 1981.

Ortega y Gasset, José. "*Misión de la Universidad*". En Paulino Garagorri *Obras de J. Ortega y Gasset*, edición nº 22. El Arquero, Alianza Editorial, 1982.

Osborne, Martin. "*The Role of Object-Oriented Technology in the Undergraduate Computer Science Curriculum*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 303-308. ACM. 1992.

Andreas Paepcke (editor). "*Object-Oriented Programming: The CLOS Perspective*". MIT Press, Cambridge (Mass.), 1993.

Pancake, Cherri M. "*The Promise and the Cost of Object Technology: A Five Years Forecast*". Communications of the ACM 38(10):32-49. October, 1995.

Parnas, David L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December, 1972.

Parnas, David L. "*Software Engineering: An Unconsummated Marriage*". Communications of the ACM 40(9): 128. September, 1997.

Parnas, David Lorge. "*Software Engineering Programs Are Not Computer Science Programs*". IEEE Software, 16(6):19-30. November/December, 1999.

Parnas, David Lorge and Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4): 259-265, December 1987.

Parrish, Allen, Lester, Cynthia, Cordes, David and Moore, Deanne. "*Assesing Computer Usage Patterns in a Software Development Course*". In Proceedings of the twenty-eighth SIGCSE technical symposium on Computer Science Education, SIGCSE '97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 58-62. ACM. 1997.

Parrish, A., Borie, R., Cordes, D., Dixon, B., Hale, D., Hale, J., Jackson, J. and Sharpe, S. "*Computer Engineering, Computer Science and Management Information Systems: Partners in a Unified Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education & Training – CSEET'98. (February 22-25, 1998. Atlanta, GA – USA). Pages 67-75. IEEE Computer Society. 1998.

Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth and Weber, Charles V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth and Weber, Charles V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July, 1993.

Paxton, John T. "*Webucation: Using the Web as Classroom Tool*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 285-289. ACM. 1996.

Pelechano Ferragud, Vicente. "*Proyecto Docente. Ingeniería del Software*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Valencia. Marzo, 2000.

Peña Marí, Ricardo. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Edición. Prentice-Hall, 1998.

Pham, Binh. "*The Changing Curriculum of Computing and Information Technology in Australia*". In Proceedings of the second Australasian conference on Computer science education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. 1997.

Piattini Velthuis, Mario G. "*Definición de una Metodología para el Desarrollo de Bases de Datos Orientadas al Objeto Fundamentadas en Extensiones del Modelo Relacional*". Tesis Doctoral. Facultad de Informática, Universidad Politécnica de Madrid. 1994.

Piattini Velthuis, Mario Gerardo. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Piattini Velthuis, Mario G., Calvo-Manzano, José A., Cervera, Joaquín y Fernández, Luis. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

Piattini Velthuis, Mario G. y Daryanani Daryanani, Sunil N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Rama, 1995.

Pozo Pardo, A. del. "*La Didáctica de Hoy*". Hijos de Santiago Rodríguez, Burgos, 1982.

Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 2nd edition. McGraw Hill, 1987.

Pressman, Roger S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

Pressman, Roger S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

Prieto, Máximo and Victory, Pablo. "*Pedagogical Pattern #20. Identity Pattern*". Version 1.0. http://www-lifia.info.unlp.edu.ar/ppp/pp20.htm. [Última vez visitado, 20/8/1999]. July, 1999.

Prieto Arambillet, Félix. "*Apuntes de la Asignatura Programación III*". Versión 1.0. Departamento de Informática. Universidad de Valladolid. Diciembre, 1999.

Principia Cybernetica Web. "*Web Dictionary of Cybernetics and Systems*". http://pespmc1.vub.ac.be/ASC. 1997.

Quillian M. R. "*Word Concepts: A Theory and Simulation of some Basic Semantic Capabilities*". Behavioural Science, 12:410-30. 1967.

Ramamoorthy, C. and Sheu, P. "*Object-Oriented Systems*". IEEE Expert, 3(3). Fall, 1988.

Rand, Ayn. "*Introduction to Objectivist Epistemology*". New American Library, 1979.

Randell, Brian. "*Memories of the NATO Software Engineering Conference*". In *Anecdotes Column*, James E. Tomayko (Editor). IEEE Annals of the History of Computing, 20(1):51-54. January-March, 1998.

Rans, Michael. "*A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development*". http://users.ox.ac.uk/~ball0370/documents/oo.pdf [Última vez visitado, 22/12/1999]. November, 1999.

Rasala, Richard. "*Design Issues in CS Education*". SIGCSE Bulletin. December, 1997.

Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

Redwine, S. T. "*Software Technology Maduration*". In Proceedings of the 1st International Conference on Software Engineering, Washington D. C. (USA). IEEE, 1985.

Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

Reynolds, Charles and Fox, Christopher. "*Requirements for a Computer Science Curriculum Emphasizing Information Technology Subject Area: Curriculum Issues*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 247-251. ACM. 1996.

Riser, Robert and Gotterbarn, Donald. "*On-line Journal: A tool for enhancing Student Journals*". In Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 203-205. ACM. 1998.

Rising, Linda. "*Design Patterns: Elements of Reusable Architectures*". Annual Review of Communications, Vol. 49:907-909. 1996.

Rivas, Erick, DeSilva, Dilhar, McDaniel, Terrie and Atkinson, Colin. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January, 1997.

Roberts, Eric, Shackelford, Russ, LeBlanc, Rich and Denning, Peter J. "*Curriculum 2001: Interim Report from the ACM/IEEE-CS Task Force*". In Proceedings of the thirtieth SIGCSE technical symposium on Computer science education, SIGCSE '99. (March 24-28, 1999, New Orleans, LA - USA). Pages 343-344. ACM. 1999.

Rodríguez Dieguez, J. "*Didáctica General*". Cincel, 1980.

Rodríguez Rubio, Miguel. "*Proyecto Docente. Análisis de Sistemas Informáticos*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Facultad de Informática. Universidad de A Coruña. Julio, 1999.

Rosson, Mary Beth and Carroll, John M. "*Scaffolded Examples for Learning Object-Oriented Design*". Communications of the ACM, 39(4):46-47. April, 1996.

Rumbaugh, James E. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). ACM. Reprinted in ACM SIGPLAN 22(12):466-481. October, 1987.

Rumbaugh, James. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March, 1994.

Rumbaugh, James. "*Building Boxes: Composite Objects*". Journal of Object-Oriented Programming (JOOP), 7(7):12-22. November-December, 1994.

Rumbaugh, James. "*OMT: The Object Model*". Journal of Object-Oriented Programming (JOOP), 7(8):21-27. January, 1995.

Rumbaugh, James. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming (JOOP), 7(9):6-12. February, 1995.

Rumbaugh, James. "*OMT: The Functional Model*". Journal of Object-Oriented Programming (JOOP), 8(1):10-14. March-April, 1995.

Rumbaugh, James. "*OMT: The Development Process*". Journal of Object-Oriented Programming (JOOP), 8(2):8-16,76. May, 1995.

Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

Rumbaugh, James. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

Rumbaugh, James. "*Depending on Collaborations: Dependencies as Contextual Associations*". Journal of Object-Oriented Programming (JOOP), 11(4):5-9. July/August, 1998.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

Russell, B. "*La Perspectiva Científica*". 2ª edición. Editorial Ariel, 1987.

Russell, B. "*Respuestas*". Edición de Lee Eisler. Península, 1997.

Sakkinen, M. "*Disciplined Inheritance*". In Proceedings of ECOOP'89. Cook, S. editor. Pages 39-56. Cambridge University Press, 1989.

Sanchís Marco, F. y Torralba Martínez, J. M. "*Seguimiento del Mercado Laboral como Guía para los Diseños Curriculares. El Caso de las Ingenierías Informáticas*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (31):18-23. 1997.

Sarle, Warren S. "*Neural Network FAQ*". June, 1996.

Savater, F. "*El Valor de Educar*". 9ª edición. Editorial Ariel, S.A., 1998.

Sawyer, Pete and Kotonya, Gerald. "*SWEBOK: Software Requirements Engineering Knowledge Area Description*". In [Abran et al., 1999], 1999.

Scragg, Greg, Baldwin, Doug and Koomen, Hans. "*Computer Science Needs and Insight-Based Curriculum*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 150-154. ACM. 1994.

Schmauch, C. "*ISO9000 for Software Developers*". IEEE Computer Society Press, 1994.

Schmerl, Bradley. "*Configuration Management and Version Control*". http://tuvalu.csflinders.edu.au/seweb/scm/lectures/cmvc1.html. 23 May 1996.

Schmucker, K. "*Object-Oriented Language for the Macintosh*". Byte, 11(8). August, 1986.

Sernadas, A., Fiadeiro, J., Sernadas, C. and Eric, H.-D. "*The Basic Building Blocks of Information Systems*". In *Information Systems Concepts*. North Holland, Namur, 1989.

Shackelford, Russell L. and LeBlanc, Richard J. "*Integrating 'Depth First' and 'Breadth First' Models of Computing Curricula*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 6-10. ACM. 1994.

Sharp, Helen, Robinson, Hugh and Woodman, Mark. "*Software Engineering: Community and Culture*". IEEE Software, 17(1):40-47. January/February, 2000.

Shaw, Mary. "*Abstraction Techniques in Modern Programming Languages*". IEEE Software, 1(4). October, 1984.

Shaw, Mary (editor). "*The Carnegie-Mellon Curriculum for Undergraduate Computer Science*". Springer-Verlag, 1985.

Shaw, Mary. "*Prospects for an Engineering Discipline of Software*". IEEE Software, 7(6):15-24. November, 1990.

Shaw, M., and Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

Shaw, Mary and Garlan, David. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996.

Shaw, Mary and Tomayko, James E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August, 1991.

Shlaer, S. and Mellor, S. "*Object Lifecycles: Modeling the World in States*". Yourdon Press, 1992.

Sierra Bravo, R. "*Tesis Doctorales y Trabajos de Investigación Científica. Metodología General de su Elaboración y su Documentación*". Paraninfo, 1986.

SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May, 1987.

Smith, M. and Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

Snyder, A. "*Inheritance in Object-Oriented Programming Languages*". In Lenzerini, M., Nardi, D. and Simi, M. editors. *Inheritances Hierarchies in Knowledge Representation and Programming Languages*. Pages 153-172. John Wiley & Sons, 1991.

Sommerville, Ian. "*Software Engineering*". 2nd edition. Addison-Wesley, 1985.

Sommerville, Ian. "*Software Engineering*". 3rd edition. Addison-Wesley, 1989.

Sommerville, Ian. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

Spivey, J. M. "*The Z Notation. A Reference Manual*". International Series in Computer Science. Prentice-Hall International, 1989.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". Technical Report HPL-94-34, Hewlett-Packard Laboratories, April 1994.

Stepanov, A. A. and Lee, M. "*The Standard Template Library*". STL Documentation. October. 1995.

Stillings, N., Feinstein, M., Garfield, J., Rissland, E., Rosenbaum, D., Weisler, S. and Baker-Ward, L. "*Cognitive Science: An Introduction*". The MIT Press, 1987.

Stroustrup, Bjarne. "*The C++ Programming Language*". 1st Edition, Addison Wesley, 1986.

Stroustrup, Bjarne. "*What Is Object-Oriented Programming?*". In Proceedings of 14th ASU Conference. August 1986.

Stroustrup, Bjarne. "*The C++ Programming Language*". 2nd Edition, Addison Wesley, 1991.

Stroustrup, Bjarne. "*The Design and Evolution of C++*". Addison Wesley, 1994.

Stroustrup, Bjarne. "*The C++ Programming Language*". 3rd Edition, Addison Wesley, 1997.

SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 16/3/2000]. February, 2000.

Sutcliffe, Alistair and Maiden, Neil. "*The Domain Theory for Requirements Engineering*". IEEE Transactions on Software Engineering, 24(3): 174-196. March, 1998.

Szyperski, Clements and Pfister, Cuno."*First International Workshop on Component-Oriented Programming WCOP'96*". 8 July 1996.

Taivalsaari, A. "*On the Notion of Inheritance*". ACM Computing Surveys, 28(3). 1996.

Tesler, L. "*Object-Oriented Dynamic Languages*". In Proceedings of the Object Expo Conference. July, 1993.

Tewari, Rajiv. "*Software Reuse and Object-Oriented Software Engineering in the Undergraduate Curriculum*". In Proceedings of the 26th SISCSE technical symposium on Computer Science Education, SIGCSE '95. (March 2-4, 1995, Nashville, TN, USA). Pages 253-257. ACM, 1995.

Tewari, Raj and Friedman, Frank. "*The Impact of Object-Oriented Software Engineering in the Introductory Computer Science Curriculum*". In addendum to the proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 289-292. ACM. 1992.

The British Computer Society and The Institution of Electrical Engineering. "*A Report on Undergraduate Curricula for Software Engineering Curricula*". June, 1989.

Thompson, S. "*Haskell, The Craft of Functional Programming*". Addison-Wesley, 1999.

Tilley, Scott, Smith, R. and Dennis B. "*Perspectives on Legacy System Reengineering*". Software Engineering Institute. Draft – Version 0.3. Carnegie Mellon University, Pittsburgh. 1995.

Tomayko, James E. "*Teaching a Project-Intensive Introduction to Software Engineering*". Technical Report CMU/SEI-87-TR-20 (ESD-TR-87-171), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August, 1987.

Tomayko, James E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

Toro Bonilla, Miguel (editor). "*Actas de las I Jornadas de Trabajo en Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 14-15 de noviembre de 1996.

Toval Álvarez, Ambrosio y Nicolás Ros, Joaquín (editores). "*Actas de las III Jornadas de Ingeniería del Software*". Departamento de Informática, Lenguajes y Sistemas de la Universidad de Murcia. Murcia, 11-13 de noviembre de 1998.

Tremblay, Guy. "*Knowledge Area Description for Design (version 0.5)*". In [Abran et al., 1999], 1999.

Tucker, Allen B. and Wegner, Peter. "*New Directions in the Introductory Computer Science Curriculum*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer science education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 11-15. ACM. 1994.

Tucker, Allen B. (Editor and Co-chair), Barnes, Bruce H. (Co-chair), Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke, Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*Computing Curricula 1991. Report of the ACM/IEEE-CS Joint Curriculum Task Force*". ACM. http://www.acm.org/education/curr91/homepage.html. December, 1990.

Tucker, Allen B., Barnes, Bruce H., Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke, Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*Computing Curricula 1991*". ACM Press. February, 1991.

Tucker, Allen B., Barnes, Bruce H., Aiken, Robert M., Barker, Keith, Bruce, Kim B., Cain, J. Thomas, Conry, Susan E., Engel, Gerald L., Epstein, Richard G., Lidtke,

Doris K., Mulder, Michael C., Rogers, Jean B., Spafford, Eugene H. and Turner, A. Joe. "*A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report. Computing Curricula 1991*". Communications of the ACM, 34(6):68-84. June, 1991.

Tucker, Allen B., Astrachan, Owen, Bruce, Kim, Cupper, Robert, Denning, Peter, Drysdale, Scot, Horton, Tom, Kelemen, Charles, McGeoch, Cathy, Patt, Yale, Proulx, Viera, Rada, Roy, Rasala, Richard, Roberts, Eric, Rudich, Steven, Stein, Lynn, Van Loan, Charles. "*Strategic Directions in Computer Science Education*". ACM Computing Surveys, 28(4):836-845. December, 1996.

Tuya González, Pablo Javier. "*Manual de Procedimientos para las Prácticas de Ingeniería del Software I y II*". Versión 1.2. Universidad de Oviedo. http://opalo.etsiig.uniovi.es/~tuya/is/procedimientos/procs.htm. [Última vez visitado, 1-12-1999]. Octubre, 1999.

Tymann, Paul T., Lea, Douglas and Raj, Rajendra K. "*Developing an Undergraduate Software Engineering Program in a Liberal Arts College*". In Proceedings of the twenty-fifth annual SIGCSE symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 276-280. ACM. 1994.

Ullman, Jef and Widom, Jennifer. "*A First Course in Database Systems*". Prentice-Hall, 1997.

UNESCO. "*Aprender a Ser*". Informe de la Comisión Internacional para el Desarrollo de la Educación. Editorial Alianza, Madrid. 1973.

Ungar, David, Smith, Randall B., Chambers, Craig and Hölzle, Urs. "*Object, Message and Performance: How they Coexist in Self*". IEEE Computer 25(10): 53-64. October, 1992.

Universidad de Salamanca. "*Guía Académica Curso 1998-1999. Facultad de Ciencias*". Facultad de Ciencias - Universidad de Salamanca. 1998.

Universidad de Salamanca. "*Guía Académica Curso 1999-2000. Facultad de Ciencias*". Facultad de Ciencias - Universidad de Salamanca. 1999.

Vaitkevitchius, Raimundas. "*Pedagogical Pattern #25. BASE-and-Supplementary-Languages in Lectures (BSLL)*". In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp25.htm. [Última vez visitado, 20/8/1999]. July, 1999.

Vaquero Sánchez, Antonio. "*La Lengua Española en el Contexto Informático*". Revista de Enseñanza y Tecnología. ADIE. (13):5-12. Enero-Abril, 1999.

Vaughn, Rayford B. Jr. "*Software Engineering Degree Programs*". Crosstalk, The Journal of Defense Software Engineering, 13(3):7-9. March, 2000.

Veraart, V. E. and Wright, S. L. "*Supporting Software Engineering Education with Local Web Site*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 275-279. ACM. 1996.

Vetter, Ronald J. and Severance, Charles. "*Web-Based Education Experiences*". IEEE Computer, 30(11):139-141. November, 1997.

Villarreal, E. E. and Butler, D. "*Giving Computer Science Students a Real-World Experience*". In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 40-44. 1998.

Vivaracho Pascual, Carlos Enrique. "*Proyecto Docente. Teoría de la Información y Codificación – Sistemas Operativos*". Área de Conocimiento de Ciencias de la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Julio, 1996.

Walker, Henry M. and Schneider, G. Michael. "*A Revised Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 39(12):85-95. December, 1996.

Wallingford, Eugene. "*Toward a First Course Based on Object-Oriented Patterns*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 27-31. ACM. 1996.

Wand, Y. "*How to Integrate Object Orientation with Structured Analysis and Design*". IEEE Software, 6(2). March, 1989.

Ward, P. and Mellor, S. "*Structured Development for Real-Time Systems*". Vols. 1-3. Yourdon Press, 1985.

Ward, Paul T. and Mellor, Stephen J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

Wegner, Peter. "*The Object-Oriented Classification Paradigm in Research Directions on Object-Oriented Programming*". MIT Press, Cambridge, MA, 1987.

Wegner, Peter. "*Concepts and Paradigms of Object-Oriented Programming*". OOPS Messenger, 1(1). August, 1990.

Weinberg, G. F. "*The Psychology of Computer Programming*". Van Nostrand Reinhold, 1971.

Welch, David and Strong, Scott. "*An Exception-Based Assertion Mechanism for C++*". Journal of Object-Oriented Programming (JOOP), 11(4):50-60. July-August, 1998.

Wielinga, B. J., Schreiber, A. T. and Breuker, J. A. "*KADS: A Modeling Approach to Knowledge Engineering*". Technical Report ESPRIT Project P5248 KAD-II, 1991.

Wirfs-Brock, Rebecca and Johnson, Ralph E. "*Surveying Current Research in Object-Oriented Design*". Communications of the ACM, 33(9):104-124. September, 1990.

Wirfs-Brock, Rebecca, Wilkerson, Brian and Wiener, Lauren. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

Wirth, Niklaus. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April, 1971.

Wirth, Niklaus and Reiser, Martin. "*Programming in Oberon — Steps Beyond Pascal and Modula*". Addison-Wesley, Reading (Mass.), 1992.

Wohlin, Claes and Regnell, Björn. "*Achieving Industrial Relevance in Software Engineering Education*". In Proceedings of the 12th Conference on Software Engineering Education and Training (CSEE&T '99). (22 - 24 March, 1999. New Orleans, Louisiana – USA). IEEE Computer Society. Pages 16-25. 1999.

Woodman, Mark, Davies, Gordon and Holland, Simon. "*The Joy of Software – Starting with Objects*". In Proceedings of the twenty-seventh SIGCSE technical symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 88-92. ACM. 1996.

Yonezawa, A. and Tokoro, M. "*Object-Oriented Concurrent Programming: An Introduction*". In *Object-Oriented Concurrent Programming*. Cambridge, MA: The MIT Press. 1987.

Yordon, E. and Constantine, L. "*Structured Design*". 1st edition. Prentice Hall, 1975.

Yordon, E. and Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

Yordon, E. and Constantine, L. "*Structured Design*". 2nd edition. Yourdon Press, 1989.

Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

Yourdon, Edward. "*Modern Structured Analysis*". Prentice Hall, 1989.

Zazo Rodríguez, Ángel Francisco. "*Proyecto Docente para el Perfil Teleinformática Aplicada a las Ciencias de la Documentación*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Facultad de Traducción y Documentación. Universidad de Salamanca. Diciembre, 1996.

Zilles, S. "*Types, Algebras, and Modeling*". In *On Conceptual Modeling: Perspectives from Artificial Inteligence, Databases, and Programming Languages*. Springer-Verlag, 1984.