

AN ANALYSIS OF SELECTED SOFTWARE SAFETY STANDARDS

Dolores R. Wallace, D. Richard Kuhn, Laura M. Ippolito
National Institute of Standards and Technology¹
Gaithersburg, MD 20899 USA

ABSTRACT

This study examines standards, draft standards, and guidelines (all of which will hereafter be referred to as documents) that provide requirements for the assurance of high integrity software. The study focuses on identifying the attributes necessary in a document for providing reasonable assurance for high integrity software, and on identifying the relative strengths and weaknesses of the documents. The documents vary widely in their requirements and the precision with which the requirements are expressed. Security documents tend to have a narrow focus and to be more product oriented, while safety documents tend to be broad in scope and center primarily on the software development process. Overall there is little relationship between the degree of risk and the rigor of applicable standards. Recommendations are provided for a base standard for the assurance of high integrity software.

INTRODUCTION

High integrity software is software that must be trusted to work dependably in some critical function, and whose failure to do so may have catastrophic results, such as serious injury, or loss of life or property [NIST190]. Examples include civil aviation, medical devices, nuclear power, weapons systems, and electronic funds transfer (EFT). While we rely on computerized systems in every aspect of life, we do not always have confidence in the software. Examples of past and potential catastrophes can be found in the annual "Risk of the Year" presentations at COMPASS (Computer Assurance) conferences and in major research reports [COMP91, USC89, NRC91].

Developers and customers need a standard framework of requirements that will provide reasonable assurance that the software of critical systems will be of high integrity. Many organizations (e.g., industry associations, international standards organizations) have been developing standards to serve this purpose. In this study we identify some of the documents that are available and their strengths and weaknesses. We develop a set of criteria to identify the key characteristics of each document, and to determine how well a given document satisfies the criteria for each characteristic.

While the documents we chose to study (see Table 1) vary widely (e.g., in scope, lifecycle coverage, quality coverage),

we were able to extract some common approaches in engineering practices and assurance requirements. From the findings, we propose a set of topics, with basic requirements, as an outline for a base document on which to develop a standard for the assurance of high integrity software (see Table 2).

PURPOSES OF STANDARDS

Standards serve many purposes. Standards serve as a yardstick for comparing systems. This is the most obvious use of standards. They provide reference points to indicate minimum acceptable requirements, or to compare systems.

International standards are becoming extremely important to economic competitiveness. In the near future (probably 1993) products will need to meet minimum quality standards to be sold in the European Community countries.

Standards are a means of specifying the requirements for a product. This is an important use of standards within the government. Agencies want to be able to state their requirements with minimum effort. The easiest way to do this is to require conformance to a standard that is accepted for their particular application. Product procurement is simplified by requiring conformance to a standard, rather than listing required product characteristics in every procurement.

Standards are a means of improving the state of practice. This third item is a non-obvious, but nevertheless important, use of standards. In fact, for some of the safety and security issues discussed in this paper this may be the most important use. Companies that are not developing software that satisfies a customer's standard have to either redirect their work or find other customers. As more and more customers require higher quality products, the state of practice tends to improve and non-competitive companies leave the industry.

Equally important to setting a minimum quality level is the fact that a standard represents a commonly agreed upon set of requirements for developers. When customers have many different definitions of what they want in their products, developers have a hard time keeping up with the varied and sometimes competing requirements. Standards make higher quality products more economical by providing common requirements.

¹Systems and Software Technology Division, Computer Systems Laboratory, Technology Administration, U.S. Department of Commerce.

Acronym	Number and Title (complete references are in Section 6)
ANS7432	ANSI/IEEE-ANS-7-4.3.2-1982: Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations
CATEGORY	Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities with respect to Nuclear Safety, Revision 0, 1991
DLP880	DLP880: (DRAFT) Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)
EWICS2-1	Dependability of Critical Computer Systems 2, Chapter 1: Guidelines to Design Computer Systems for Safety, 1989
EWICS2-2	Dependability of Critical Computer Systems 2, Chapter 2: Guidelines for the Assessment of the Safety and Reliability of Critical Computer Systems, 1989
EWICS2-3	Dependability of Critical Computer Systems 2, Chapter 3: A Questionnaire for system Safety and Reliability Assessment, 1989
EWICS2-4	Dependability of Critical Computer Systems 2, Chapter 4: A Guideline on Software Quality Assurance and Measures, 1989
EWICS2-5	Dependability of Critical Computer Systems 2, Chapter 5: Guidelines on the Maintenance and Modification of Safety-Related Computer Systems, 1989
IDS0055	Interim Defence Standard 00-55: The Procurement of Safety Critical Software in Defence Equipment, 1991
IEC880	IEC 880: Software for Computers in the Safety Systems of Nuclear Power Stations, 1986
IECSUPP	45A/WG-A3(Secretary)42: (DRAFT) Software for Computers Important to Safety for Nuclear Power Plant, 1991
NPR6300	NPR-STD-6300: Management of Scientific, Engineering and Plant Software, 1991
P1228	P1228: (DRAFT) Standard for Software Safety Plans (IEEE Working Group), 1991
RTCA178A	RTCA/DO-178A: Software Considerations in Airborne Systems and Equipment Certification, 1985
SOFTENG	Standard for Software Engineering of Safety Critical Software, Rev. 0, 1990

Table 1. Documents in the Study

STUDY OF STANDARDS

We have examined many of the current standards, draft standards, draft revisions, and guidelines, which may be national, international or industry-specific, that address requirements for software assurance. In general we selected documents whose purpose is to provide requirements for safety critical software and eliminated those providing fundamental requirements for quality in general purpose software. We examined in detail a subset of those documents, listed in Table 1; these will be referred to by the acronyms in Table 1. We focused on documents from the nuclear industry since this application area obviously requires high integrity software; we elected to concentrate on those documents to understand how well, collectively, they may provide assurance

for high integrity software, but we also included several documents intended for large critical systems. While we may extend the detailed study to include all the documents available to us, this selection provides a reasonable sample of current requirements.

We are interested in the following two questions for each document: Does the document provide reasonable assurance of high integrity software? Can the customer determine that the developer has met the requirements of the document?

We developed a set of topics (shown in Table 2) for examining each document. For these topics we identified detailed criteria necessary for a reasonable standard; we derived many of the detailed criteria from specific standards

Levels of Criticality/Assurance
Lifecycle Phases
Documentation
Required Functionality
Engineering Practices
Assurance Activities
Software Verification and Validation (V&V)
Software Quality Assurance (SQA)
Software Configuration Management (SCM)
Hazard Analysis
Project Planning and Management
Procurement Concerns
Presentation
Security/Software Safety Issues

Table 2. Criteria Template

(e.g., IEEE1012 and IEEE828). Table 3 lists reference documents used in deriving the criteria and the acronyms by which we refer to them in this paper.

We did not expect a single document to fully address every criterion; rather our purpose was to determine how well a document satisfies requirements for any criterion it claimed to cover. Our intention is to understand how well the best available guidance from each document might collectively support the assurance of high integrity software.

Levels of Criticality/Assurance

Some documents have established software requirements based on the consequence of system failure. The most serious consequence is usually considered to be loss of life and is assigned the highest level of criticality. Other levels of criticality take into account how serious a failure is relative to the completion of the task the system is responsible for and how devastating the failure may be relative to destruction of property and environment, injuries, and other losses. For the most critical systems, the most rigorous software standards and practices must be invoked. We looked for distinctions in a document's requirements based on levels of criticality. This includes requirements according to levels of criticality not only for the principal system but for its support software and for software used to develop and assure it.

Lifecycle Phases

Some documents were developed for the system lifecycle while the scope of others began with the development of

software requirements and did not fully address integration of software within the total system. For comparison purposes, we used only the software lifecycle, and looked for activities related to requirements, design, code, integration and test, installation, and maintenance and operation. We use the phases only to identify the scope of each document. We are not concerned whether a document specifies a particular lifecycle management (e.g., waterfall, spiral). While some activities in software development and assurance may need to be performed at certain times in the lifecycle (e.g., system test planning during requirements), in general we do not make judgments on lifecycle management. Our priorities center on the activities themselves and how well a document addresses a particular phase. By identifying documents that address partial lifecycles, it may become clear which document, or arts of them, may be used together.

In documents dealing with the system lifecycle it is sometimes difficult to know when a requirement is imposed on the software, or takes effect only after integration of the software with system components (e.g., configuration management). We focused on software related activities. We also checked that the documents made a clear distinction between software related activities and system related activities.

Documentation

While we concentrated on *software* documentation, in some cases it was necessary to consider a document's requirements for the system requirements specifications because those specifications levy requirements for software. We considered the following types of questions:

Acronym	Number and Title (complete references are in Section 6)
ANS104	ANSI/ANS-10.4-1987: Guidelines for the Software Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry
ASMENQA2	ASME NQA-2a-1990: Quality Assurance Requirements for Nuclear Facility Applications
DOT86	Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use, 1986
FIPS101	FIPS 101: Guideline for Lifecycle Validation, Verification, and Testing of Computer Software, 1983
FIPS132	FIPS 132: Guidelines for Software Verification and Validation Plans, 1987
FIPS1401	FIPS 140-1: Security Requirements for Cryptographic Modules, 1990
IEEE828	ANSI/IEEE Std 828-1983: IEEE Standard for Software Configuration Management Plans
IEEE1012	ANSI/IEEE Std 1012-1986: IEEE Standard for Software Verification and Validation Plans
IEEE1058	ANSI/IEEE Std 1058.1-1987: IEEE Standard for Software Project Management Plans
IEEE7301	ANSI/IEEE Std 730.1-1989: IEEE Standard for Software Quality Assurance Plans
ISO9000	ISO 9000: International Standards for Quality Management, 1990
ITSEC	ITSEC 1.1989: Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems
NIST180	NIST Special Publication 500-180: Guide to Software Acceptance, 1990
NIST190	NIST Special Publication 500-190: Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991
SAFEIT	SafeIT, 1990
TCSEC	DOD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria

Table 3. Reference Documents of the Study

- How thorough are the document's requirements for specific documentation?
- Does the document specify the content that must be described in the documentation? Or, does it specify the description of elements of the content?
- Does the document provide a quantified description of attributes that should be present in the documentation (e.g., rules for maintainability, consistency)?
- Is a checklist included?
- Do the requirements for a document specify required functionality or engineering practices?

Required Functionality

Critical systems must continue to operate despite errors and component failures. To help assure this, special functions are often included to detect, tolerate, override or recover from failures, or to prevent execution of unintended functions. Special functions should be a consideration in a standard for safety critical software. Examples include functions that save dynamic memory to allow restart in the event of a system crash, and use of redundancy for disk storage. Such functions

should be a consideration in a standard for safety critical software. Not all of the functions listed in the template are essential in all systems, but an evaluator should look for the use of these functions, or for reasons why they are not needed in a particular system.

Engineering Practices

A standard for safety critical software should give guidance on engineering practices that contribute to high integrity. Certain engineering practices can either contribute or detract from the safety and reliability of a system. For example, systems constructed from modules that each perform a single, well-defined function are likely to be more reliable than those where modules perform a mixture of functions (e.g., both control and data input/output). Choice of programming language is another example [NRC91]. Systems written in assembly language tend to be much more error-prone than those developed in modern high-level languages such as Pascal, C, and Ada [NRC91].

Rigid rules for the use of specific engineering practices are not always appropriate. In some cases, it may not be possible to develop software for a particular application in a way that is normally considered good engineering practice in other applications. For example, it is considered good practice to separate critical functions from the rest of the system, placing critical functions in a small module that can be more readily analyzed than a large system. It has been argued that sometimes safety critical systems cannot be built in this way. In those systems, safety concerns are present in all functions. It is essential that developers use established, good practices as much as possible, and explain cases where established practices were not used.

Guidance for engineering practices should be considered in all projects, even if not all techniques are appropriate for all projects. The major types of engineering practices considered in this review were: formal specifications; critical component isolation; modularity; high order languages; deprecated programming practices (e.g., floating point arithmetic); and, quality attributes with quantified definition.

Assurance Activities

Assurance activities locate problems in the development processes and products and provide evidence on how well the software complies with its specifications. The activities include software verification and validation (V&V), software quality assurance (SQA), software configuration management (SCM), and hazard analysis. Many of the documents we examined addressed the system lifecycle; in our study we needed to be careful whether system or software activities were addressed. For example, system configuration management may well mean that software configuration management during software development is not required.

Two standards together have comprehensive requirements for software V&V. These are ANSI104 and FIPS132, which references IEEE1012. The criteria for software V&V are based on these standards. The criteria for SQA are derived from IEEE7301. This standard relies heavily on the existence of project documentation. While the thrust of SQA is *product* assurance, process assurance and process improvement are also important (see Section 4.6.2). While we mention these topics under "Procurement Concerns," they are outside the scope of our study. The criteria for SCM are taken from IEEE828, a standard which is presently serving as a foundation document for an ISO/IEC JTC1 SC7 (Software Engineering) working group for developing a standard on SCM.

A prerequisite for any critical system is an analysis of the hazards or threats that the system must protect against. For example, a power plant safety shutdown system must continue to function even during a power failure. While we were mostly concerned with hazard analysis applied to software, it should be noted that software hazard analysis (e.g., software fault tree analysis) is an integral part of system hazard analysis, and that both should be conducted to assure that all hazards have been identified.

Project Planning and Management

Well-defined project management procedures are as important for the development of high integrity software as they are for any quality product. The documents we reviewed are broad in scope and should contain some requirements for how the development of software will be planned, managed, and monitored. While the detailed criteria are sparse, we also considered requirements from IEEE1058 and guidelines developed for another Federal agency.

Procurement Concerns

Some evolving standards address concerns of the customer. For example, the customer of a system may have some concerns about the people building and evaluating the system. Are they capable? Should evaluators be independent of the vendor? What should their training plans look like? Do the companies have a quality management policy? Some standards also address assessment of qualifications of the vendor and vendor plans to remain qualified. Another procurement issue involves the use of automated support to build and verify the system. Any topics included in the documents that affect the customer at contract are discussed in this section of the template.

Presentation

One of the major problems with using a standard and verifying compliance with it is that all too often the "requirements" of the standard are specified in a disorderly,

ambiguous manner. Different categories of requirements are often specified in documentation requirements. For example, we prefer to see required functionality and required engineering practices stated separately rather than in the documentation requirements for a specific process (e.g., design).

Security/Safety Issues

Software safety has much in common with computer security [NRC91]. Both fields are concerned with preventing certain undesirable events. Software safety seeks to avoid accidents that present a hazard to life or property. Security is concerned with both accidents and malicious attacks that violate an organization's security policy. Given this similarity of concerns, it seems natural that safety and security standards might be very much alike. In comparing the safety documents reviewed in this study with security standards [TCSEC, ITSEC, FIPS1401, DOT86], we found significant contrasts in presentation, level of detail, and approaches to assurance.

FINDINGS OF THE STUDY

Levels of Criticality/Assurance

Among the documents reviewed for this report, most have addressed levels of criticality. Canadian documents DLP880 and SOFTENG do not cite levels of criticality because DLP880 implicitly assumes it is addressing critical software, and SOFTENG is for safety-critical systems (the highest level of requirements). The purpose of the third Canadian document, CATEGORY, is to provide guidance on classifying software according to the consequences of failure, but it does not associate software engineering practices with those categories.

IEC880 makes no distinctions concerning assurance needs in the main sections of the document. While Appendix B of IEC880 identifies recommendations for design and programming practices by three levels of priority or importance, no guidance is given on a definition of priority or importance. One purpose of IECSUPP is to clarify and supplement IEC880; perhaps the final IECSUPP will provide guidance on levels of criticality. The draft refers only to specifying diversity requirements (e.g., hardware or software functional requirements), depending on reliability requirements. While ANS7432 neglects to address levels of assurance, it does require that the tools used for verification must have quality assurance activities performed on them commensurate with their importance to the verification process. IDS0055 is written for the highest level of criticality and very strongly states that all support tools for safety critical software must be at the same assurance level as that warranted by results of hazard analysis on them relative to the safety-critical software.

EWICS2-1 states that the design constraints should be associated with the level of criticality. In EWICS2-2, seven levels of criticality are related to types of systems and values for attributes like unavailability and failure probability. While the questionnaire in EWICS2-3 provides a table of complexity factors, including the scope of safety considerations, the outcome is not associated with specific practices or assurance techniques. NPR6300 classifies software according to the hardware, function, or activity it supports.

RTCA178A discusses three levels of criticality, and the levels must be addressed in the certification plan for the software. RTCA178A is currently undergoing revision; draft version 4 defines five levels of criticality and provides guidance for determining the level of a system. The international computer security community addresses seven levels of assurance [TCSEC, ITSEC].

There is some disagreement within the software engineering community with respect to levels of assurance and the requirements that are appropriate for each level. The central question concerns whether we know enough to say which assurance methods provide greater confidence in system safety.

Lifecycle Phases

While we identify phases in this study, our intention is to determine whether a document has requirements for necessary processes, including those for assurance. Some documents are special purpose documents and address those parts of the lifecycle phases that they affect. For example, the categorization document, CATEGORY, is concerned with procedures and guidelines for categorizing software; the categorization is assigned at the initiation of a project. EWICS2-1 provides guidelines for the design of safety critical systems and does not provide guidance on other lifecycle issues. The primary concern in EWICS2-1 is the process of design at the system level with some guidance on software considerations.

Some of the documents deal strictly with requirements for the software lifecycle, such as IEC880, IECSUPP, SOFTENG, and RTCA178A. However, they do place the software phases in context with system phases. Other documents, like DLP880 and ANS7432 which address safety systems, also address system requirements and integration of hardware and software. The EWICS2 documents are concerned at the system level, with some specific references to software. In many of the documents, there is often confusion as to whether system or software is the focus of activity.

For safety systems in which software is embedded and must always be related to the system, the following issues on the lifecycle are especially important:

- Does the document relate software activities to system requirements?
- By treating software as part of the system, does the document remove necessary emphasis on software (e.g., configuration management requirements at the system level only)?
- Are all lifecycle processes covered well by combining requirements of the documents?

We found that no single document provides sufficient requirements for the first two questions. The combination of the documents provides coverage, at least minimally, in the lifecycle processes. While IEC880, in spite of its problems with presentation, comes close, it does not address project management. Other documents do better in other areas (e.g., NPR6300 on reuse and corrective action). While ANS7432 does address the software-system relationship, overall its requirements for software for all lifecycle processes are either minimal or nonexistent. If taken as a whole, the EWICS2 documents address (system) design and maintenance, but provide little guidance on other aspects of the software lifecycle processes. EWICS2-4 and SOFTENG are the only two documents which address quality attributes and measures for them. With respect to maintenance, IEC880 and EWICS2-5 provide more guidance than the other documents.

RTCA178A provides rather generic requirements for most lifecycle processes. It is currently under revision; the revision will probably be oriented more toward processes, not phases, and may contain rigorous requirements.

Documentation

The requirements for documentation range from a simple statement for each type of document to a complete description of the quality attributes of a document. ANS7432 falls into the terse category (e.g., completeness, consistency, and documentation standards are implied). The most complex set of documentation requirements are those in DLP880 for documentation to be written in formal specification languages. IDS0055 also has strong requirements for formal specification languages.

Only one document, SOFTENG, provides rigorous guidance on the quality attributes that should be inherent in documentation. For each document, criteria are identified for each required quality attribute.

One of the features of several documents, especially IEC880 and SOFTENG, is that documentation requirements include requirements for the software itself. For example, in IEC880 documentation requirements specify that system modules should be designed with a single well-defined function. The design of modules with a single well-defined function is an intellectual activity that designers use to structure the system

for the best possible design for the system's operational capability and assurance; the primary purpose of implementing this practice is not documentation. The documentation should reflect, not require, the design resulting from this intellectual activity. In other cases, functional requirements were hidden in documentation requirements. We recommend that standards make the distinction between documentation requirements and those for engineering practices and functionality; this distinction should facilitate the task of demonstrating compliance with a standard. The documentation requirement may be that the engineering practices should be documented separately (e.g., IDS0055's requirements for a "Code of Design Practices").

In most of the reviewed documents, separate documentation is specified for each lifecycle phase or process. An exception is RTCA178A which treats the software development and verification plan as one document.

Required Functionality

IEC880 and EWICS2-3 include lists of functions that can be used to counter specific hazards. Of these two, the checklists in EWICS2-3 are the more comprehensive. IEC880 contains annotations indicating what each function is "good for" and "good against," but these are generally obvious, so the annotations provide little useful guidance. For example, the annotation for Retry Procedures indicates that retries are useful against sporadic hardware faults; range checking of variables is said to help guard against "yet undetected errors."

The inclusion of checklists of functions to guard against hazards is helpful in a standard, but it is probably not appropriate to mandate specific functions when a standard covers a broad category of systems. Some functionality that is considered essential for safety systems (e.g., range checking) might be required, but in general, some functions may not be appropriate for all systems. This is consistent with documents for high integrity systems supporting security. Documents for general purpose secure systems have less prescribed functionality than those for a more narrow range of applications, such as data encryption.

Engineering Practices

Engineering practices are techniques that help prevent errors during system construction, or help ensure integrity in operation. An example of the first type is the use of formal specification languages, and an example of the second type is the use of modularity.

The documents reviewed vary enormously in their recommendations regarding engineering practices. Most contain at least some guidance on good practices with the two exceptions of RTCA178A and NPR6300. While EWICS2-4 and EWICS2-5 do not address engineering practices, the other EWICS2 documents provide comprehensive coverage of

recommended practices. The summaries below are given approximately in order of consensus among the standards.

Modularity and critical component isolation - The engineering practices cited by most of the documents are the use of modularity in design and the isolation of critical components.

Programming language - Several documents state principles for programming languages. The consensus is for use of high level languages (e.g., C, Fortran, Ada) rather than assembler, and for languages that support automatic checking of data types and function arguments. For example, Ada or C++ will warn if a function is called with an integer argument when it is expecting a character string. Another programming consideration is the use of structured programming (the use of restricted control structures rather than arbitrary branching).

Formal methods - Over the past decade, there has been increasing interest in the use of formal methods, i.e., the use of mathematical logic and related areas of mathematics to specify and model the behavior of software. Formal methods are required by only two of the documents reviewed (DLP880, IDS0055). Another, the supplement to IEC880 (IECSUPP), says that "formal methods should be considered for the highest requirement of safety importance." EWICS2-3 gives preference to the use of formal specifications over informal ones, but does not require the use of formal methods. IEC880 notes only that "a formal specification language may be a help to show coherence and completeness of the software functional requirements." A trend toward greater reliance on formal methods is evident in the documents we reviewed. DLP880, IDS0055 and IECSUPP were 1991 drafts; EWICS2-3 was written in 1989, and IEC880 in 1986. In the fields of computer security and data communications, formal methods are becoming accepted practice, and are required at the higher levels of all significant security standards.

Documentation of engineering practices - The documents reviewed gave adequate treatment to most aspects of engineering practices, except in the area of formal methods. (This statement reflects the documents in general. As noted, there was much variation.)

Quality attributes - Only SOFTENG and EWICS2-4 provided either specific requirements or measures for quality attributes like completeness, consistency, and maintainability.

Assurance Activities

In many industries, software is one component of a company's business. At the top management level, the view is of the whole, not a part. System configuration management and system validation are the engineering concepts that make sense to executives of manufacturing companies. For software companies, executives think in terms of software configuration management and software validation. The difference is non-trivial and has caused much misunderstanding in the

development of standards. Software is deeply embedded in systems where software cannot fully stand alone. In these systems, often non-software components are not only plug-in but are built to precise, accredited standards. Configuration management and testing of these components during their development are expected activities. Software should be treated similarly. To date this has not been possible for testing. First, software systems are usually unique for each system in which they will be embedded. There may be no precise set of validated specifications. Second, their full functionality can only be simulated and tested in real-time.

In this study we concentrated on how well the documents provide assurance of the software. Few of the documents are focussed entirely on software. We found ourselves second-guessing if system level requirements are applied at the software development level, or only at the point when software is integrated with the system. If we are unsure of requirements, how can auditors check for exact compliance? If more accredited, precise standards for software existed, as they do for other components (e.g., pipes, power cables), then the task of demonstrating compliance would be easy. This study reemphasized the growing recognition that the software industry must develop more precise standards for software that permit measurement of its quality.

For the assurance activities, P1228 focuses on safety issues and requires specific assurance activities. Under P1228, all documentation for assurance activities may serve as special sections of plans for those activities (e.g., safety requirements for the software V&V plan, the SQA plan, the SCM plan). The assumption of the P1228 draft is that the other IEEE Software Engineering Standards, or similar standards, will be used.

Software Verification and Validation (V&V). The difference between system and software viewpoints stands out in the documents. For example, ANS7432 is concerned with computer system validation and not particularly concerned with software issues. Software testing comprises software verification; in the software world this can be confusing because software verification also includes many types of static analysis. Part of the rationale for not treating verification and validation as separate functions in IEEE1012 is to avoid this confusion. The final step of software V&V is the system validation, as in system standards; software V&V consists of these activities applied as the software evolves to assure the internal properties of the software and the external relationships to the system.

DLP880 refers to software verification but is in reality software V&V. One caution with DLP880 is the assumption that the vendor may produce the verification plan which is then implemented by an independent team. One should not think this is the only meaning of independent V&V (IV&V), because the fullest possible benefit of independence is the independent planning process in which the IV&V brings a

different perspective to the types of analysis and test strategies.

Two documents, EWICS2-1 and P1228, focus on the safety requirements; this is acceptable since these documents are intended to augment other more general documents and the intent is to ensure attention to the safety functions. EWICS2-3 should be used by verifiers to guide them in checking features of the software, and auditors to check how well the developers and verifiers have followed guidelines.

IEC880 specifically addresses software verification, and is reasonably thorough. There are weaknesses, however. The major weakness is that of presentation; a reader has to search several places before finding all the requirements for a given process, in this instance, verification. Technical weaknesses include a lack of specific requirements for requirements traceability.

While some test strategies are recommended in several documents, some major strategies have been omitted. For example, IEC880 has long lists of strategies and conditions but omits stress testing. Error analysis should be a requirement in all V&V standards or sections of standards addressing V&V (or possibly in SQA). Error analysis is important for uncovering a type of error (e.g., misunderstanding of trigonometry) that could appear elsewhere in the system. When the type of error is made because of a misunderstanding or a wrong specification, it is important to check other places in the program that are based on the same assumptions, especially if the same person is responsible. Otherwise, a potentially critical error could slip through.

While RTCA178A provides a well organized set of requirements for software verification (including validation), the software verification assurance matrix is too high-level to be truly useful for auditors.

From our review of these documents, including the base documents IEEE1012 and ANS104, recommendations for improving standards for software V&V include the following:

- clearer relationship to the system lifecycle
- practices based on levels of criticality
- distinct requirements for different types of test
- detailed checklists
- application of V&V when modern development technologies are used (e.g., no document addressed V&V for prototyping or expert systems)
- error analysis

- definition of the quality attributes for which verification is required.

Software Quality Assurance (SQA). The documents of this study are concerned primarily with SQA of the product, not the vendor processes. Current and evolving SQA standards are addressing process as well as product. When a customer audits a particular product, will the customer be concerned about whether a vendor has changed processes mid-stream, or for the next product? Probably not. For a current audit, the customer will likely be interested in whether a given product has the required quality level. But, when new SQA standards are written, what happens if they require activities for both process and product? The revision of RTCA178A may address process quality. Customers of high integrity software need to study this question to determine if process quality is outside the scope of their auditors.

Several documents address SQA in a general manner. For example, ANS7432 requires that SQA be addressed in the software development plan. IEC880 simply requires an SQA plan. This is insufficient because it will not be clear what is required of the vendors; for audit purposes, the customer must know the following:

- the minimum set of SQA activities that are to be performed
- to what degree SCM and software V&V are included in SQA.

Some standards permit national standards to be used. Each user of a standard needs to ensure specification of an SQA standard in a contract or expect its auditor to know every SQA standard quite well.

Design and code inspections can be either SQA or software V&V activities. Requirements for inspections were not consistent in the documents. Both ANS7432 and EWICS2-1 provide detailed procedures for SQA of design, and EWICS2-4 addresses SQA entirely.

There is a growing recognition that SQA procedures are needed for existing software programs and for reuse of software modules. NPR6300 provides detailed guidance, and it appears that IECSUPP will address the topic also. The SQA sections of the documents, like those for software V&V, are in general weak concerning anomaly reporting, corrective action and follow-up, and error analysis.

Software Configuration Management (SCM). Software configuration management is another process that sometimes is addressed only at the system level. While the size of some software systems used in safety critical applications may be small, the critical role of software for safety mandates that

SCM be required for all the lifecycle processes and products of such software. IEC880's requirement for system configuration management is insufficient. ANS7432 and EWICS2-3 simply ignore the topic. Several documents, DLP880, SOFTENG, RTCA178A, and NPR6300, require SCM activities with varying degrees of rigor. The international community (ISO/IEC JTC1 SC7) is presently using IEEE828 as a base document for producing an international standard on SCM. Perhaps the completed international standard could serve as the basis for SCM in a standard for assurance of high integrity software.

Hazard Analysis. While typically the initial hazard analysis is performed from a total system, environmental perspective, the results of that hazard analysis and successive hazard analyses must be studied to identify whether software has any role in the potential occurrence of those hazards occurring or in mitigating the effects of those hazards. Some traditional engineering methods for hazard analysis are being applied at the software level.

The purpose of CATEGORY is to identify the criticality category of software for the nuclear industry. This document does not define methods for hazard analysis but does provide criteria for criticality to be applied to the results of hazard analysis.

Many of the documents appear to assume that a hazard analysis/criticality assessment will have been performed (e.g., IEC880, DLP880). SOFTENG and RTCA178A indirectly require hazard analysis. P1228 requires safety-related analyses on the results of the system preliminary hazard analysis; the guidance in this draft standard should be required for software related to large complex systems.

Project Planning and Management

Most of the documents in this study either do not address project management activities or do so indirectly through other governing principles. For example, requirements on planning for SQA and SCM may be considered requirements for project planning. SOFTENG includes requirements for project management in requirements for the software development plan. EWICS2-4 addresses project management through acceptance criteria. The P1228 draft expects a project management plan, and requires that the plan be augmented to address software safety issues.

Procurement Concerns

One concern that customers have is whether or not assurance activities should be performed by independent teams. Some documents, like ANS7432, use non-binding statements in the foreword to recommend independence. Others, like DLP880 and IEC880, recommend that verification plans be written so that an independent team may implement the plan. EWICS2-1 and EWICS2-2 do ask for IV&V assessment. However,

total independence means that the assessor also must prepare the V&V plan. IDS0055 requires the customer to have a special officer for safety critical software.

Some of the documents specify contractor capability assessment. The EWICS2 documents ask for compliance with ISO9000 which requires assessment of the contractor's quality system. P1228 requires the software safety plan to specify qualifications for the personnel performing software safety activities. IDS0055 requires that the qualifications of the project staff be demonstrated.

Most of the documents imply that tailoring is permitted by using "should" instead of "shall" (see Section 4.9). However, SOFTENG does have a statement that all requirements of the standard must be met for compliance. The permission to tailor a standard to a project may present two problems. One, the tailored version may not produce a satisfactory assurance of the product. Two, an auditor may have difficulty assessing compliance.

Presentation

The documents in this study have a variety of problems with their presentation. The major problem lies with the usage of words to indicate requirements: "shall," "should," "must," "may." When the words "shall" and "should" appear in the same paragraph, it can be confusing to vendors, assurers, customers, and auditors because "shall" in a standard means "required" but "should" means "desirable but not required." Requirements and recommendations need to be clearly distinctive from one another. Some requirements in several documents (e.g., "the software must be easy to test" [IEC880]) are meaningless because they are not precise enough to objectively test for conformance. An example of language with meaningful constraints on qualities may be found in SOFTENG (e.g., "quantified definition of completeness for a software requirements document").

Another concern is that features required to be present in the software, development practices, and descriptions of the software are often specified in requirements documentation. A mechanism frequently used is to define documentation as a description of a lifecycle process. For example, in IEC880 and SOFTENG documentation for a software requirements specification often specifies the principles or functions the system must embody, or in the case of SQA, the activities to be performed. Engineering practices are hidden in documentation requirements. Those practices are discussed under "Documentation" in this report. We recommend that standards separate categories of requirements.

Security/Software Safety Issues

Many systems can be considered both safety-critical and security-critical. Command and control and weapons systems are obvious examples of life critical systems with both

properties. Medical information systems are another. Since safety is defined to include property, EFT systems are also included in this category. Although safety and security concerns may overlap, existing standards have been developed from either a security or safety viewpoint, but not both. Security standards that may apply to systems with safety concerns include TCSEC, ITSEC, DOT86, and FIPS1401.

Because software safety and computer security often appear in the same system, and because the problem in both cases involves avoiding undesired events, one might expect standards for the two areas to be similar. Since we are familiar with several computer security standards, we were able to make some informal comparisons with the software safety documents examined in this study. We found some interesting contrasts between the approaches taken by the computer security and software safety communities.

Assurance can be gained in different ways. One way is to carefully control the development process to reduce the errors that may be introduced. Another approach is to require that the software product has specific features to ensure that it behaves correctly in operation, e.g., fault tolerance features. Normally both approaches must be used if one is to be confident of the resulting software, but different applications may stress process and product assurance requirements to different degrees.

Safety documents are more process oriented. The safety documents tend to concentrate on the development process, what kinds of hazard analysis must be done, how requirements tracing should be handled, and what documentation should be provided. When the safety standards do have something to say about particular features, such as error checking, it is usually treated as something that is recommended or suggested, rather than required.

Security documents are more product oriented. The security documents identify features that must be in the product, such as what kinds of error detection must be used, and what kinds of controls must be provided. They also have requirements on the development process to ensure sound engineering, but they have more details about the products than safety documents.

Security documents tend to be mandatory and more specific. The emphasis is on response to specific threats. That is, the threats tend to be assumed, and the document designed to counter those particular threats. For example, TCSEC is designed to meet the threat of people obtaining access to information for which they do not have appropriate security clearances. Having a particular set of threats as a starting point is an advantage in that the standard can be more specific in its requirements. The disadvantage is that it limits the class of systems to which the standard can be applied. For example, TCSEC is often regarded as of limited use for commercial systems because its specificity limits it to defense systems.

The security documents tend to have more development requirements for software which are more stringent. When looking at computer security and software standards together, there seems to be little relationship between the degree of risk involved in an application and the rigor of the applicable standards. It is very unlikely that a failure in an EFT system could result in anyone's death, yet the standards applicable to EFT [DOT86, FIPS1401] have more rigorous requirements (e.g., strict modularity, self testing functions, formal methods) than some for civil aviation and nuclear power.

The biggest difference between the security and safety document seems to be their basic assumptions. The safety documents concentrate on how to identify hazards, instead of assuming a particular set of threats or hazards and suggesting how to respond to them. This is probably because the class of hazards in safety critical systems is typically larger and more varied than the class of threats in security. There is an almost unlimited number of things that can go wrong when the computer is a component of a system with many interdependent and interacting pieces, and experience has shown that most safety failures result from unexpected interactions. Security has, until recently, focused on the security of individual systems with limited interfaces to the outside world. The threats to such systems were regarded as restricted to the information system itself.

With security increasingly being regarded as including integrity and dependable service in addition to confidentiality, and with the proliferation of distributed systems, secure systems will need to deal with the kinds of unexpected interactions between systems that cause problems in safety. Thus the problems of safety and security may become more similar.

All the techniques specified in safety documents have counterparts in security documents, although different names may be used. This suggests that a standard set of techniques could be applied to both areas. There are some things required by security documents, though, that have no counterpart in safety (except at a very high level of abstraction); for example, covert channel analysis. Determining the correct combination of techniques must of course vary with the application.

CONCLUSIONS and RECOMMENDATIONS

No single standard or guideline in this study completely satisfies the evaluation criteria given in Table 2. In almost every case, each document satisfies at least one category. It is important to develop a standard with rigorous requirements for engineering practices, required functionality, software project management, documentation, hazard analysis, software V&V, SQA, and SCM. The language of the standard must not be ambiguous. It is acceptable to cite specific standards to provide requirements for any of the categories (e.g., IEEE1012 and ANSI104 for software V&V).

We recommend taking the strong parts of these documents to build a rigorous guide for assurance of high integrity software, or, alternatively, to encourage development of a standard based on these documents. For example, DLP880 and IDS0055 have the most rigorous requirements for engineering practices. Requirements can be extracted from IEC880 for almost every category. P1228 provides requirements for ensuring that safety consideration are given attention at every step of development and assurance; the same concepts could be used for ensuring attention to computer security issues.

Information from all of the documents can be used in developing a rigorous standard but other concerns must be addressed. One is that the scope of the standard must be clearly identified relative to system and software lifecycles. The requirements for software must ensure that the software is always assured in relation to the system. A standard should either define a practice or specify a standard for it; citing good "software quality" practice is not sufficient because good software quality practice can be interpreted in many ways. Until software engineering practice is rigorously codified in handbooks such as in other engineering fields, we recommend that guidance for assurance of high integrity software either describe all practices or cite specific acceptable standards for them, including at least the topics in Table 2. We plan to identify the requirements for a comprehensive standard for the assurance of high integrity. We will ask the experts attending our next workshop to critique these requirements.

REFERENCES

ANS7432

ANSI/IEEE-ANS-7-4.3.2-1982, "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations," American Nuclear Society, 1982.

ANS104

ANSI/ANS-10.4-1987, "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry," American Nuclear Society, May 13, 1987.

ASMENQA2

ASME NQA-2a-1990, "Quality Assurance Requirements for Nuclear Facility Applications," The American Society of Mechanical Engineers, November 1990.

CATEGORY

"Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities with respect to Nuclear Safety," Revision 0, Nuclear Safety Department, June 1991.

COMP91

Neumann, Peter R., "The Computer-Related Risk of the Year," COMPASS '91, Proceedings of the Sixth Annual Conference on Computer Assurance, The Institute of Electrical and Electronics Engineers, Inc., June, 1991.

DLP880

DLP880, "(DRAFT) Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)," David L. Parnas, Queen's University, Kingston, Ontario, March, 1991.

DOT86

"Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use," United States Department of the Treasury, September 1, 1986.

EWICS2-1

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 1, "Guidelines to Design Computer Systems for Safety," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-2

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 2, "Guidelines for the Assessment of the Safety and Reliability of Critical Computer Systems," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-3

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 3, "A Questionnaire for System Safety and Reliability Assessment," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-4

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 4, "A Guideline on software quality Assurance and Measures," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

- EWICS2-5
Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 5, "Guidelines on the Maintenance and Modification of Safety-Related Computer Systems," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.
- FIPS101
FIPS 101, "Guideline for Lifecycle Validation, Verification, and Testing of Computer Software," U.S. Department of Commerce/National Bureau of Standards, 1983 June 6.
- FIPS132
FIPS 132, "Guideline for Software Verification and Validation Plans," U.S. Department of Commerce/National Bureau of Standards, 1987 November 19.
- FIPS1401
FIPS 140-1, "Security Requirements for Cryptographic Modules," U.S. Department of Commerce/National Institute of Standards and Technology, 1990 May 2.
- IDS0055
Interim Defence Standard 00-55, "The Procurement of Safety Critical Software in Defence Equipment," Parts 1 and 2, Ministry of Defence, 5 April 1991.
- IEC880
IEC 880, "Software for Computers in the Safety Systems of Nuclear Power Stations," International Electrotechnical Commission, 1986.
- IECSUPP
45A/WG-A3(Secretary)42, "(DRAFT) Software for Computers Important to Safety for Nuclear Power Plants as a Supplement to IEC Publication 880," International Electrotechnical Commission Technical Committee: Nuclear Instrumentation, Subcommittee 45A: Reactor Instrumentation, Working Group A3: Data Transmission and Processing Systems, May 1991.
- IEEE828
ANSI/IEEE Std 828-1983, "IEEE Standard for Software Configuration Management Plans," The Institute of Electrical and Electronics Engineers, Inc., 1983.
- IEEE830
ANSI/IEEE Std 830-1984, "IEEE Standard for Software Requirements Specifications," The Institute of Electrical and Electronics Engineers, Inc., 1984.
- IEEE983
ANSI/IEEE Std 983-1986, "IEEE Standard for Software Quality Assurance Planning," The Institute of Electrical and Electronics Engineers, Inc., 1986.
- IEEE1012
ANSI/IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," The Institute of Electrical and Electronics Engineers, Inc., November 14, 1986.
- IEEE1058
ANSI/IEEE Std 1058.1-1987, "IEEE Standard for Software Project Management Plans," The Institute of Electrical and Electronics Engineers, Inc., 1988.
- IEEE7301
ANSI/IEEE Std 730.1-1989, "IEEE Standard for Software Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., October 10, 1989.
- ISO9000
ISO 9000, "International Standards for Quality Management," May 1990.
- ITSEC
ITSEC 1.1989, "Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems," GISA - German Information Security Agency, 1989.
- NIST180
NIST Special Publication 500-180, "Guide to Software Acceptance," U.S. Department of Commerce/National Institute of Standards and Technology, April 1990.
- NIST190
NIST Special Publication 500-190, "Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991," U.S. Department of Commerce/National Institute of Standards and Technology, August 1991.
- NPR6300
NPR-STD-6300, "Management of Scientific, Engineering and Plant Software," Office of New Production Reactors, U.S. Department of Energy, March 1991.
- NRC91
"Computers at Risk," National Research Council, National Academy Press, 1991.

P1228

P1228, "(DRAFT) Standard for Software Safety Plans (IEEE Working Group)," The Institute of Electrical and Electronics Engineers, Inc., July 19, 1991.

RTCA178A

RTCA/DO-178A, "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, March 1985.

SAFEIT

"SafeIT," Volumes 1 and 2, Interdepartmental Committee on Software Engineering, ICSE Secretariat, Department of Trade and Industry, London SW1E6SW, UK, June 1990.

SOFTENG

"Standard for Software Engineering of Safety Critical Software," Rev. 0, Ontario Hydro, December 1990.

USC89

U.S. Congress, House, Committee on Science, Space, and Technology, "Bugs in the Program," U.S. Government Printing Office, September 1989.

TCSEC

DOD 5200.28-STD, "Department of Defense Trusted Computer System Evaluation Criteria," Department of Defense, December 1985.