



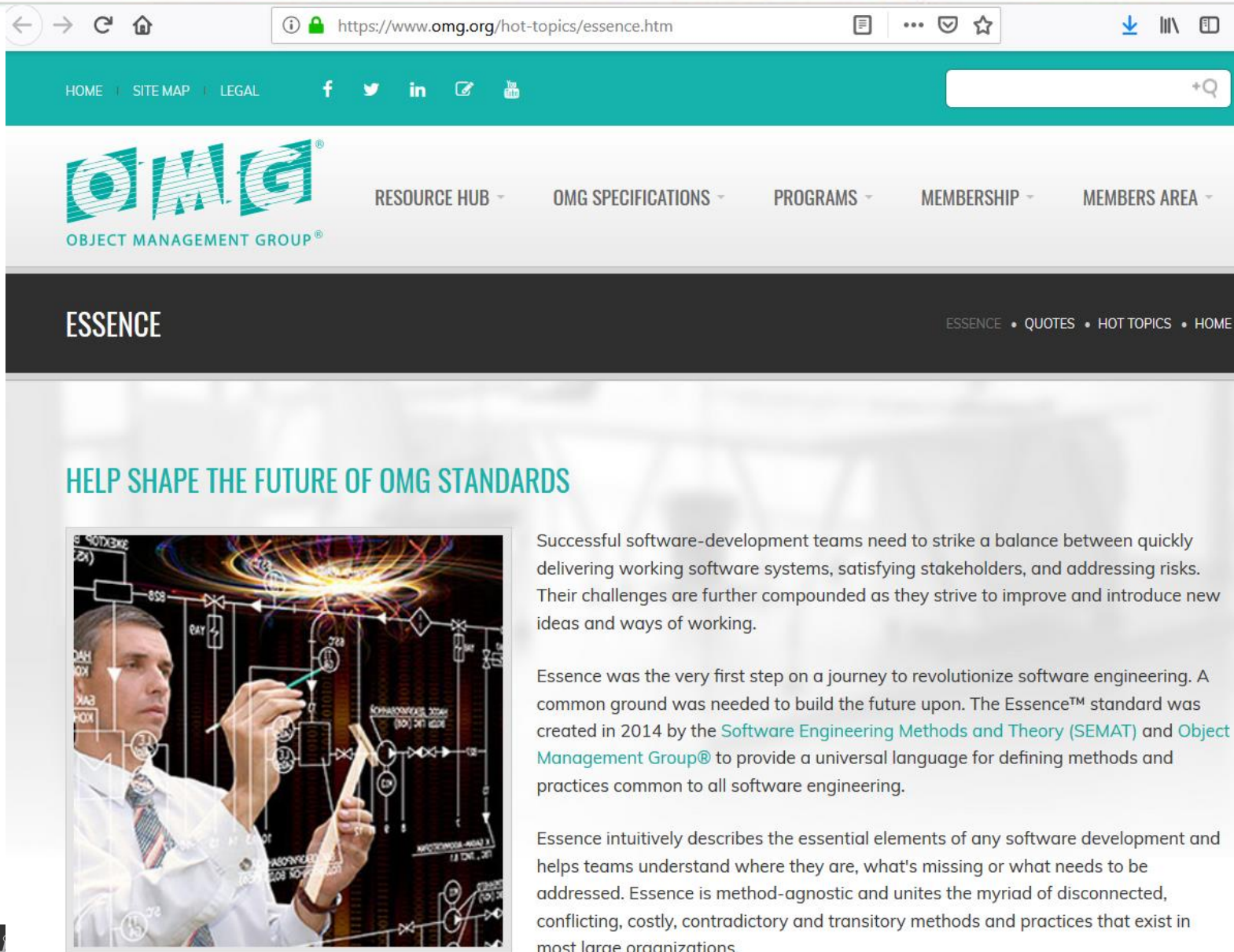
**Essence Seminar
For University of Rome "Tor Vergata"**

Essence: Software Engineering Essentialized

Giuseppe Calavaro, Ph.D.

www.semat.org

Essence on OMG web site



HOME | SITE MAP | LEGAL

f t in

OMG
OBJECT MANAGEMENT GROUP®

RESOURCE HUB ▾ | OMG SPECIFICATIONS ▾ | PROGRAMS ▾ | MEMBERSHIP ▾ | MEMBERS AREA ▾

ESSENCE

ESSENCE • QUOTES • HOT TOPICS • HOME

HELP SHAPE THE FUTURE OF OMG STANDARDS



Successful software-development teams need to strike a balance between quickly delivering working software systems, satisfying stakeholders, and addressing risks. Their challenges are further compounded as they strive to improve and introduce new ideas and ways of working.

Essence was the very first step on a journey to revolutionize software engineering. A common ground was needed to build the future upon. The Essence™ standard was created in 2014 by the [Software Engineering Methods and Theory \(SEMAT\)](#) and [Object Management Group®](#) to provide a universal language for defining methods and practices common to all software engineering.

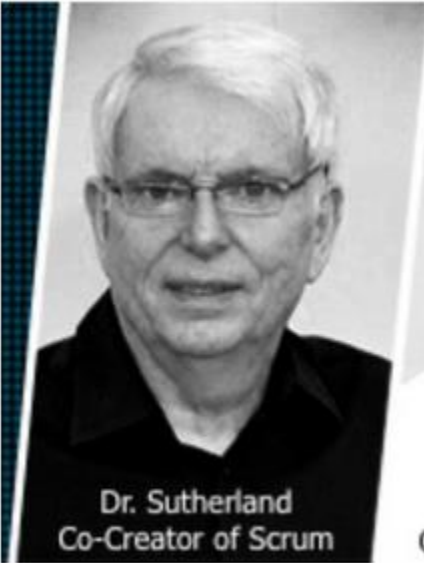
Essence intuitively describes the essential elements of any software development and helps teams understand where they are, what's missing or what needs to be addressed. Essence is method-agnostic and unites the myriad of disconnected, conflicting, costly, contradictory and transitory methods and practices that exist in most large organizations.

Essence on Semat Web site


semat.org 67%

SEMAT Home » What is SEMAT? » SCRUM & ESSENCE UNITE Essence Standard » News » Sign In

Home /



Dr. Sutherland
Co-Creator of Scrum



Dr. Jacobson
Co-Creator of Essence

SCRUM & ESSENCE PERFECT PARTNERS

Read more

Agenda of this Seminar

Software Engineering concepts are given for granted

- Introduction to *Essence*
- The Basics of Software Engineering
- The *Essence* Language
- The *Essence* Kernel
- Conclusions: Reflections on Essence Goals and Theory of Software Engineering
- Appendix: A summary of *Essence* Concepts and Elements



**Essence Seminar
For University of Rome "Tor Vergata"**

Introduction to *Essence*

Giuseppe Calavaro, Ph.D.
IBM Big Data Practice Leader
External Professor at University of Rome "Tor Vergata"

www.semat.org

What is “Essence”?



- **Essence is the kernel of a Software Engineering Theory as well as the language to describe such theory and the approach to describe methods and practices based on the theory**
 - It is available as an OMG specification and supported by a growing body of educational and other supporting material, in particular, a set of games, e.g., to identify the status of a project or to identify next steps.
 - Essence has been designed from the beginning as an educational tool and to allow students and practitioners alike to explore software processes with the help of clearly defined, easy-to-understand concepts and the support of the kernel

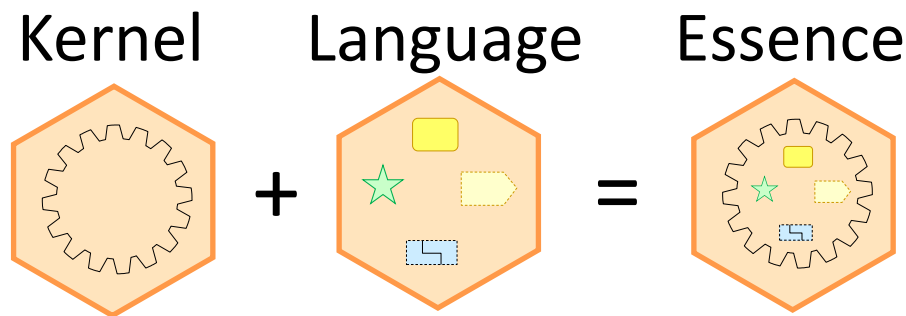
The Essence of our goal

1. Methods are compositions of practices
 - A practice, that is like a mini-method, is a reusable approach to a well defined problem
 - Practice examples: Requirements Management, Agile Development, Use Case Modeling, etc.
2. There is a common ground – Kernel- shared by all methods and practices
 - A common vocabulary
 - It makes easier to teach, learn, use and modify practices
 - Is a necessity to create a library of reusable practices from which selecting
3. Focus on the essentials when providing guidelines for a method or practice
 - Developers rarely have the time to read detailed methods and practices
 - The essential are defined as the initial minimum of what expert knows, but enough to start practicing. 5% could be enough and provide the idea that is really the essence of the whole.

The Essence of "Essence"



- Essence is made of 2 parts:
 - Kernel
 - The kernel of Software Engineering
 - The set of elements that would always be found in all types of software system endeavours
 - Language
 - The Essence language is very simple, intuitive and practical
 - Utilized in describing the Essence kernel with the elements that constitute a common ground



Essentialize Practices and Methods

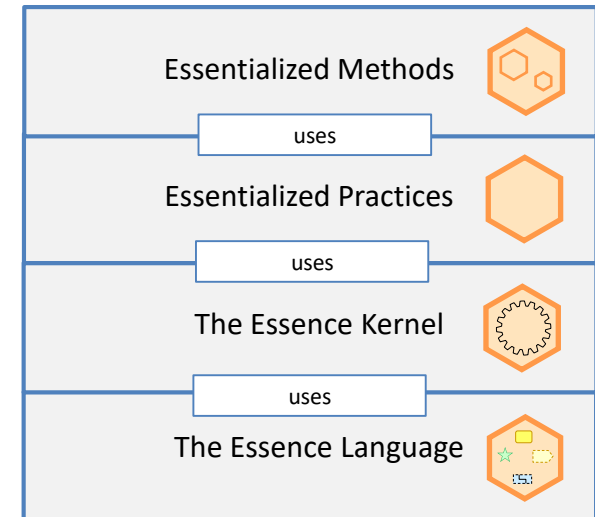


- Using Essence Kernel and Language you can Essentialize Practices and Methods
- Essentialize a practices means they are described using Essence
 - the Essence kernel
 - the Essence language
 - It focuses the description of the method/practice on what is essential.
- Consequently, the methods we describe are also essentialized.

Essence Method Architecture



- **Essentialized Methods** are composition of **Essentialized practices**
- Practices can be compositions of smaller practices.
 - Scrum for instance can be seen as a composition of three smaller practices:
 - Daily Stand-up,
 - Backlog-Driven Development
 - Retrospective.



Methods are composition of Practices

- Composition of practices is an operation merging two or more practices to form a method.
 - The operation has been defined mathematically in order to be precise.
 - The operation has to be specified by an expert with the objective to resolve potential overlaps and conflicts between the practices concerned, if there are any.
 - Usually most practices can be composed easily by setting them side by side because there are no overlaps and conflicts, but in some cases these have to be taken care of.

Resolving overlaps and conflicts

- While practices are separate, they are not independent
 - They are not like components which have interfaces over which communication will happen.
- Practices can share elements

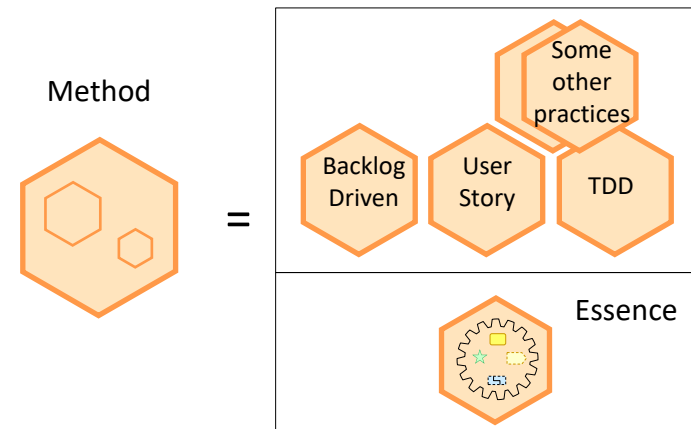
For example:

 - Guidelines for activities that a user (e.g. a developer) is supposed to perform
 - Guidelines for work products (e.g. components) that a user is expected to produce.
- If two practices share the same work product:
 - They contribute separate guidelines to this work product
 - Composing these two practices will require that you specify how the contributions must be combined in a meaningful and constructive way.

How to learn a Method



- Essence, beside text books, wants to provide team members with a new engaging and hands-on experience to learn the tailored method that each organization will define
 - A set of icons will help represent the elements
 - A set of cards will help describe and discuss the elements
- To build a method, a team start with the kernel and selects a number of practices and tools to make up its way-of-working



Cards make Kernel and Practices Tangible



- The Kernel can be “touched” and used through the use of cards
- The cards provide concise reminders and cues for team members
- By providing practical check lists and prompts, the kernel becomes something the team uses on daily basis if needed





**Essence Seminar
For University of Rome "Tor Vergata"**

The Basics of Software Engineering

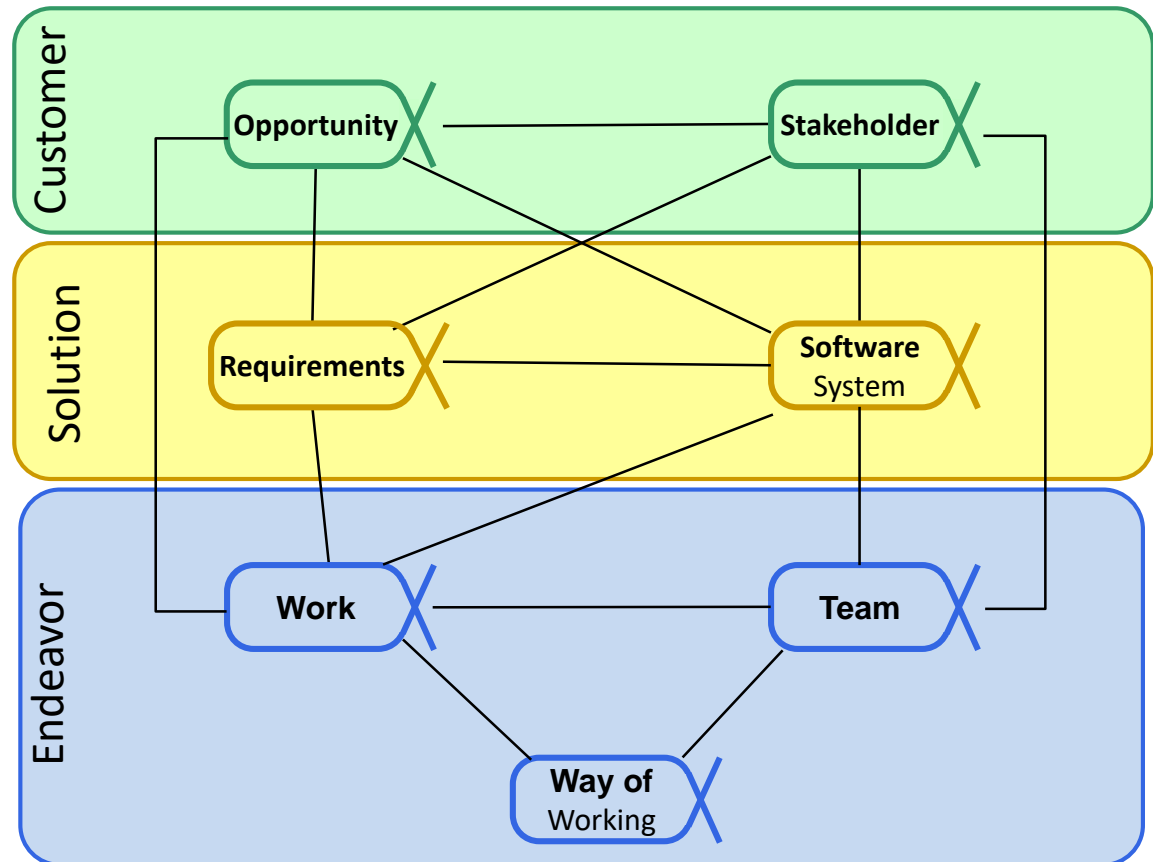
Giuseppe Calavaro, Ph.D.
IBM Big Data Practice Leader
External Professor at University of Rome "Tor Vergata"

www.semat.org

Software Engineering Basics



- The commonly used terms in Software Engineering we must know before drilling down on building methods are organized around **three areas**
- Customer
 - Opportunity
 - Stakeholder
- Solution
 - Requirements
 - Software System
- Endeavour
 - Work
 - Team
 - Way of Working



Customer

Customers: Users of our system or people that are purchasing this system for the users

Software Engineering is about providing value to customers

- Opportunity
 - An opportunity is a chance to do something to provide value to customers, including fixing an existing problem via this software system
- Stakeholders
 - Stakeholders are individuals, organizations or groups that have some interest or concern either in the system to be developed or in its development

The solution is the outcome of this endeavor

- Requirements
 - Requirements provide the stakeholder view of what they expect the software system to provide
 - They indicate what the software system must do, but do not explicitly express how it must do it
 - Among the biggest challenges software teams faces are changing requirements
- Software System
 - The primary outcome of a software endeavour is of course the software system itself.
 - 3 important characteristics of software systems
 - Functionality – Must serve some function
 - Quality – Reliability, Performance, Rich user experience, etc.
 - Extensibility – From version to version and platform to platform

Endeavors

An endeavor is any action that we take to achieve an objective

- Team
 - Team must have enough people (and not too much), with right skill mix, work collaboratively, and adapting to changing environments
 - Good team working is essential
- Work
 - The work of bringing the opportunity to reality
 - Effort and Time are the most important measures of the work
 - Effort and Time are limited
 - The idea is to get things done fast but with high quality
- Way of Working
 - Team members must agree on their way of working
 - The practice and tools that will be used
 - Used by all team members
 - Improved by the team when needed
 - One of the things we hope to achieve with Essence is simplifying the process of reaching a common agreement, that is always a major challenge



**Essence Seminar
For University of Rome "Tor Vergata"**

The *Essence* Language

Giuseppe Calavaro, Ph.D.
IBM Big Data Practice Leader
External Professor at University of Rome "Tor Vergata"





www.semat.org

Essence Prime

- Essence provides a precise and actionable language to describe software engineering practices.
 - The constructs in the Essence language are in the form of shapes and icons.
 - The different shapes and icons each have different meaning.
- Essence categorizes the shapes and icons as:
 - Things to Work With
 - Things to Do
 - Competencies
- Essence provides explicit and actionable guidance.
 - This actionable guidance is delivered through associated checklists and guidelines.

Essence Language Element Types

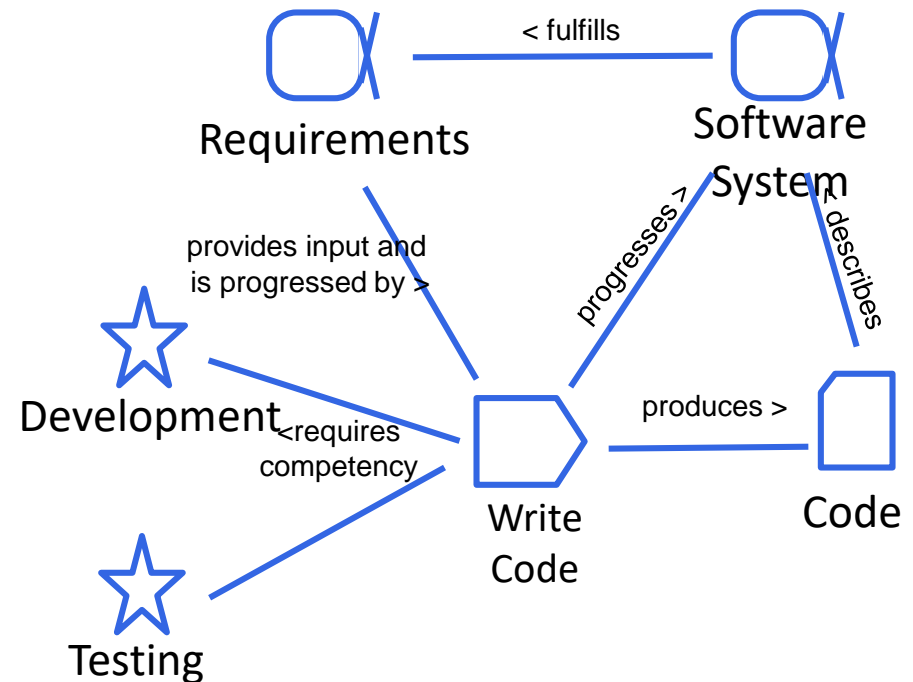


Alpha		An essential element of the software engineering endeavor that is relevant to an assessment of the progress and health of the endeavor
Work Product		The tangible things that practitioners produce when conducting software engineering activities
Activity		Things which practitioners do
Competency		Encompasses the abilities, capabilities, attainments, knowledge, and skills necessary to do a certain kind of work.

- The Essence list is longer, but at this time we consider these elements as key and the first to learn

An Example: Programming Practice

- The purpose of this practice is to produce high quality code.
 - In this case, we define code quality as being understandable by the different members of the team.
- Two persons (students) work in pairs to turn requirements into a software system by writing code together.
- Writing code is part of implementing the system.






Def: ***Alphas*** are subjects in a software endeavour whose evolution we want to understand, monitor, direct, and control

- Alphas are the most important things you must attend to and progress in order to be successful in a software development endeavour
- For our programming practice example:
 - The Alphas are: Requirements and Software System
 - There will always be requirements, regardless of whether you document them or not, or how you document them, e.g. as requirement items, use cases, etc.
 - In some cases the requirements for a software system may just exist in the heads of people. However, an alpha may be made evidenced by providing one or more descriptions; that is, by attaching work products to the alpha.
- An Alpha is not tangible by itself, but it is understood or evidenced by the work product(s) that are associated with it and thus describe a particular aspect of the alpha

Alpha Card



The Alpha Card provide a short and crisp description of the Alpha and it's States

 **Requirements**

What the software system must do to address the opportunity and satisfy the stakeholders.

Conceived


Bounded

Coherent

Acceptable

Addressed

Fulfilled

 Generated by IJI Practice Workbench™ 1.1.1

Alpha Name

Very brief
alpha description

Alpha states
Each alpha state has
an alpha state card



- Checklists are an important and practical way to monitor and guide progress
- Checklist criteria are intentionally not expressed formally
 - So teams can interpret each checklist item as they deem it appropriate to their endeavour.
- At the bottom of the card there is a bar indicating the sequence number and the total number of alpha states for this alpha

Requirements

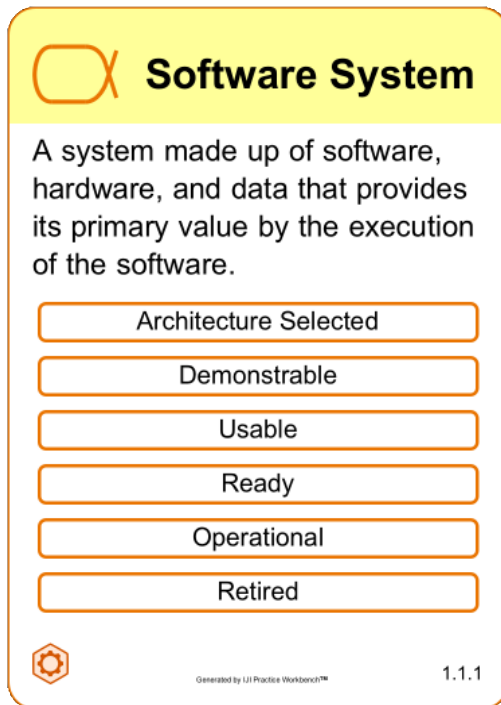
Fulfilled

- Stakeholders accept requirements
- No hindering requirements
- Requirements fully satisfied

6 / 6

Software System Alpha example

The Programming Practice in our example has also the Software System Alpha



The states are defined on the basis of an incremental risk driven approach to building the Software System:

- **Architecture Selected** – key decisions about the Software System have been made.
 - For instance, the most important system elements and their interfaces are agreed upon.
- **Demonstrated** – key use of the Software System has been demonstrated and agreed.
- **Useable** – the Software System is usable from the point of view of its users.
- **Ready** – the Software System has sufficient quality for deployment to production, and the production environment is ready.
- **Operational** – the Software System is operating well in the production environment.
- **Retired** – the Software System is retired and replaced by a new version of the Software System, or by a separate Software System.




Def: ***Work Products*** are tangible things such as documents and reports

- Work products may provide evidence to verify the achievement of alpha states.
 - For example, when a complete and accepted requirements document has been developed that evidence can be used to confirm achieving certain checklists within a state of the Requirements alpha.
- The fact that you have a document is not necessarily a sufficient condition to prove evidence of state achievement.
 - Historically, documentation has not always provided an accurate measurement of progress.
 - It is the checklist for that state that has been achieved satisfactorily the condition to satisfy
- Essence does not specify which work products are to be developed
 - But it does specify
 - What work products are,
 - How you represent them
 - What you can do with them

Work Product Card




**Code**


Good code that not only implements requirements, but also in a self-explanatory way.

Pseudo Coded

Code Completed

Code Explained

Describes:  Software System

Generated by IJI Practice Workbench™

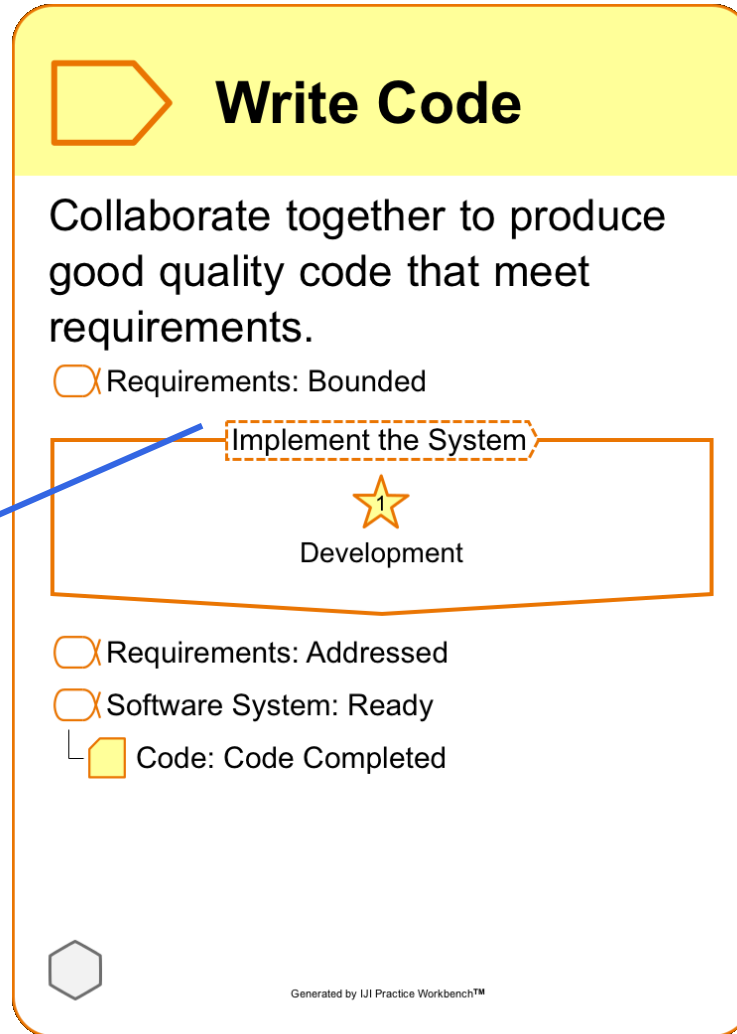
- Work Product Name
- Brief work product description
- Level of detail
- Relationship to other elements



Def: ***Activities*** are things which practitioners do

- Activities examples are: holding a meeting, analysing a requirement, writing code, testing or peer review
- Practitioners often struggle to determine the appropriate degree of detail or formality with an activity, or exactly how to go about conducting the activity.
 - This is another motivation for explicit practices as they can provide guidance to practitioners in selecting appropriate activities as well as provide guidance in how to go about conducting each activity.
- A practice may include several activities that are specific to the practice being described.
 - Activities are specific and not standard – they are not a part of Essence.
- An activity is always bound to a specific practice, it cannot “float around” among the practices.
 - If you find an activity that needs to be reused by many practices, then you may want to create a separate practice including this activity.

Activity Card



Activity space
which this activity
belongs to

Activity name

Very brief
Activity description

Inputs for activity

Competency to
conduct activity

Outputs of activity

Competency




Def: ***Competencies*** are the abilities needed when applying a practice






- Often software teams struggle because they don't have all the abilities needed for the task they have been given.
 - In these situations, a coach can help by explaining different ways the practitioner can address the problem, such as learning something that is missing in their competencies.
 - A useful exercise that teams are encouraged to conduct is to do a self-assessment of their competencies and compare the results to the competencies they believe they need to accomplish their specific endeavour.


Competency Card



**Development**

The ability to design and program effective software systems following the standards and norms agreed by the team.

Innovates	
Adapts	
Masters	
Applies	
Assists	



Generated by IJL Practice Workbench™

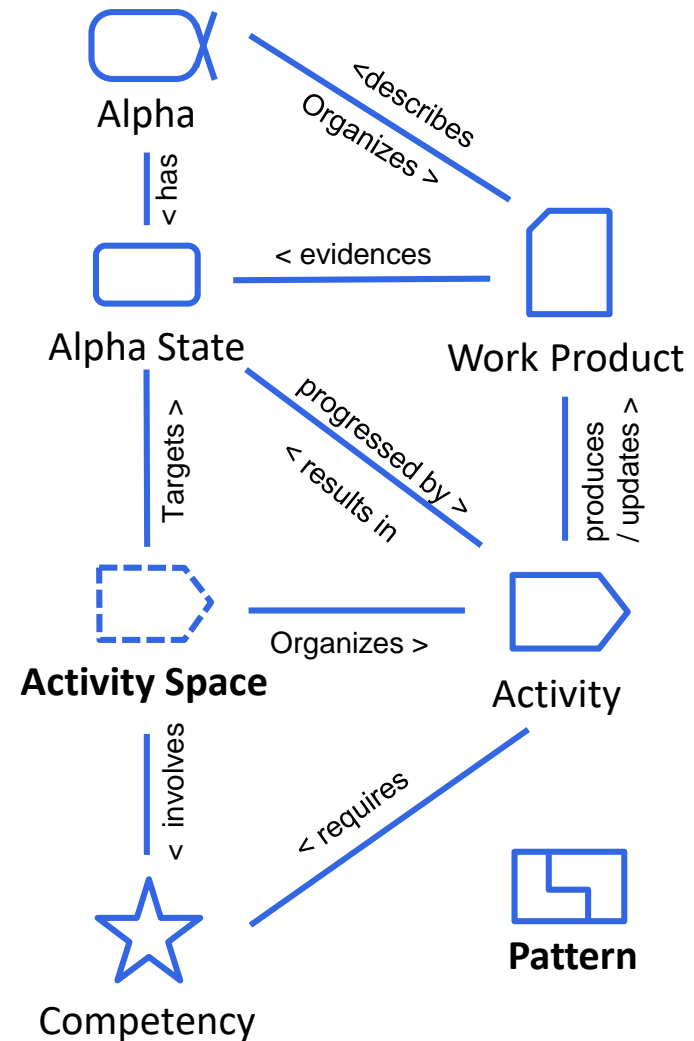
Competency name

Brief
Competency
description

Competency levels



Essence Language

- These are the elements of ESSENCE LANGUAGE and their relationships
- Essentializing a Practice, means to describe a practice using the Essence language.



Additional Elements in Essence Language

- The Essence Language Element list contains two more elements

Element Type	Syntax	Meaning of element type
Activity Space		A placeholder for something to do in the software engineering endeavor. A placeholder may consist of zero to many activities.
Pattern		An arrangement of other elements represented in the language.

- These elements will be described deeper later on.

Essentializing a Practice

- The steps to Essentializing a practice are:
 - ***Identifying the elements*** – this is primarily identifying a list of elements that make up a practice.
 - The output is essentially a diagram like that one seen for the Programming Practice
 - **Drafting the relationships** between the elements and the outline of each element
 - At this point, the cards are created
 - **Providing further details** – Usually, the cards will be supplemented with additional guidelines, hints and tips, examples, and references to other resources, such as articles and books.

Please Note: Alphas vs Products difference

Essence distinguishes between elements of health and progress versus elements of documentation.

- Elements of health and progress: Alphas
 - Alphas are the important things we work with when conducting software engineering activities.
 - Alphas are not work products.
 - Alphas are things we want to track the progress of.
- Elements of documentation: products.
 - Work products are tangible things such as documents, which can have *different levels of detail*.



**Essence Seminar
For University of Rome "Tor Vergata"**

The *Essence* Kernel

Giuseppe Calavaro, Ph.D.
IBM Big Data Practice Leader
External Professor at University of Rome "Tor Vergata"

www.semat.org

The Essence Kernel



- The Essence kernel is the set of Essence elements that would always be found in all types of software system endeavours.
 - For instance, the element architecture was discussed as a kernel element.
 - The opinion was that while for many systems it is critical to identify an architecture there are many simpler systems where architecture is not an issue.
 - Since it is not common to all projects, architecture is not a concern that every endeavor has to face, it didn't qualify as a kernel element.
- In the following slides we will illustrate the elements that are part of Essence Kernel

Areas of Concerns

- The Essence kernel elements are organized around 3 areas of concerns, that we have already seen:

Customer – This area of concern contains everything to do with the actual use and exploitation of the software system to be produced.

Solution - This area of concern contains everything related to the specification and development of the software system.

Endeavor - This area of concern contains everything related to the development team and the way that they approach their work

The Essence Kernel



The kernel elements are fundamentally of four kinds:

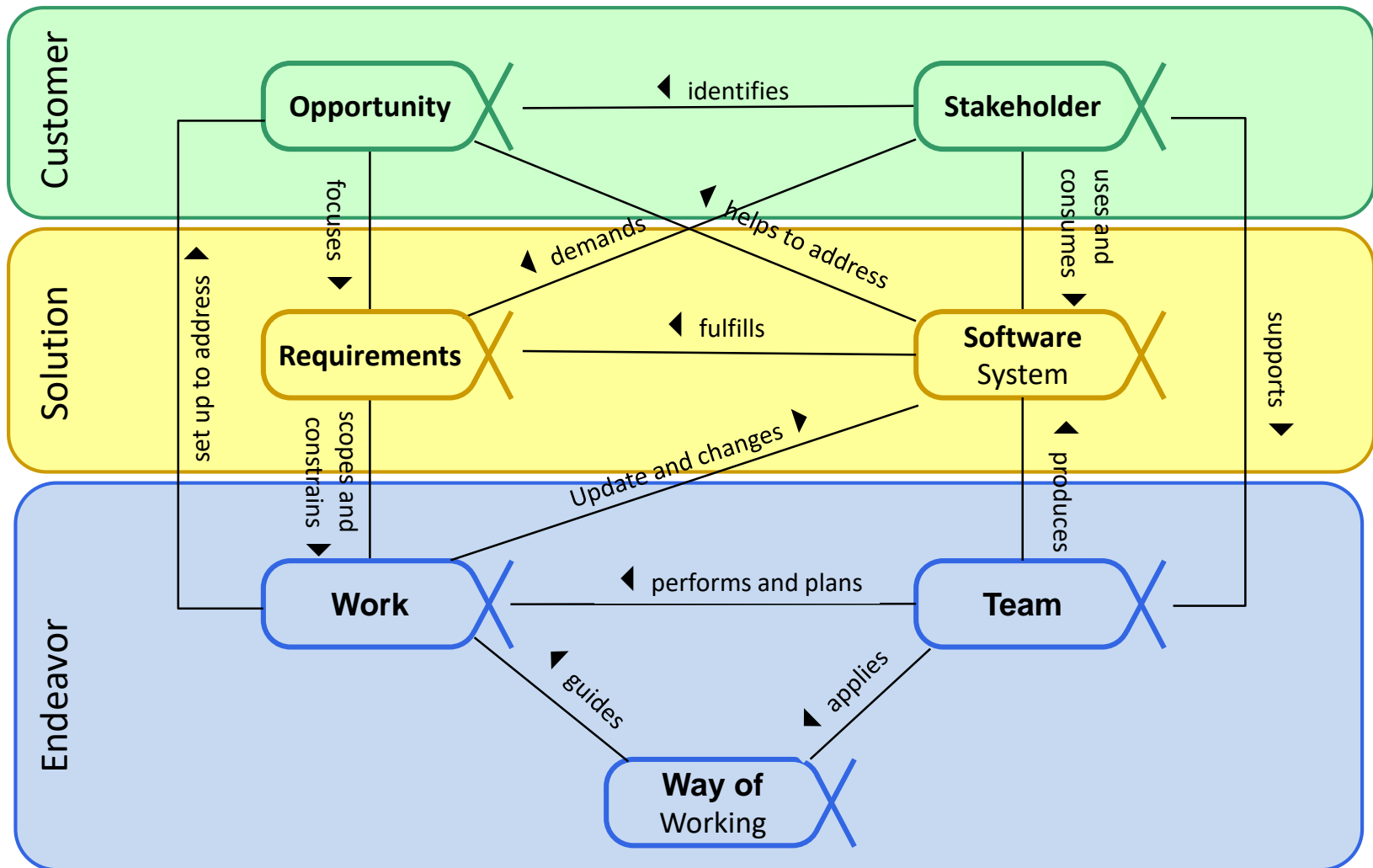
1. The essential things to work with – the alphas
2. The essential things to do – the activity spaces
3. The essential capabilities needed – the competencies
4. The essential arrangements of elements – the patterns.

- Finding the right elements is crucial.
- They must be universally acceptable, significant, relevant and guided by the notion that,
“You have achieved perfection not when there is nothing left to add, but when there is nothing left to take away.”*

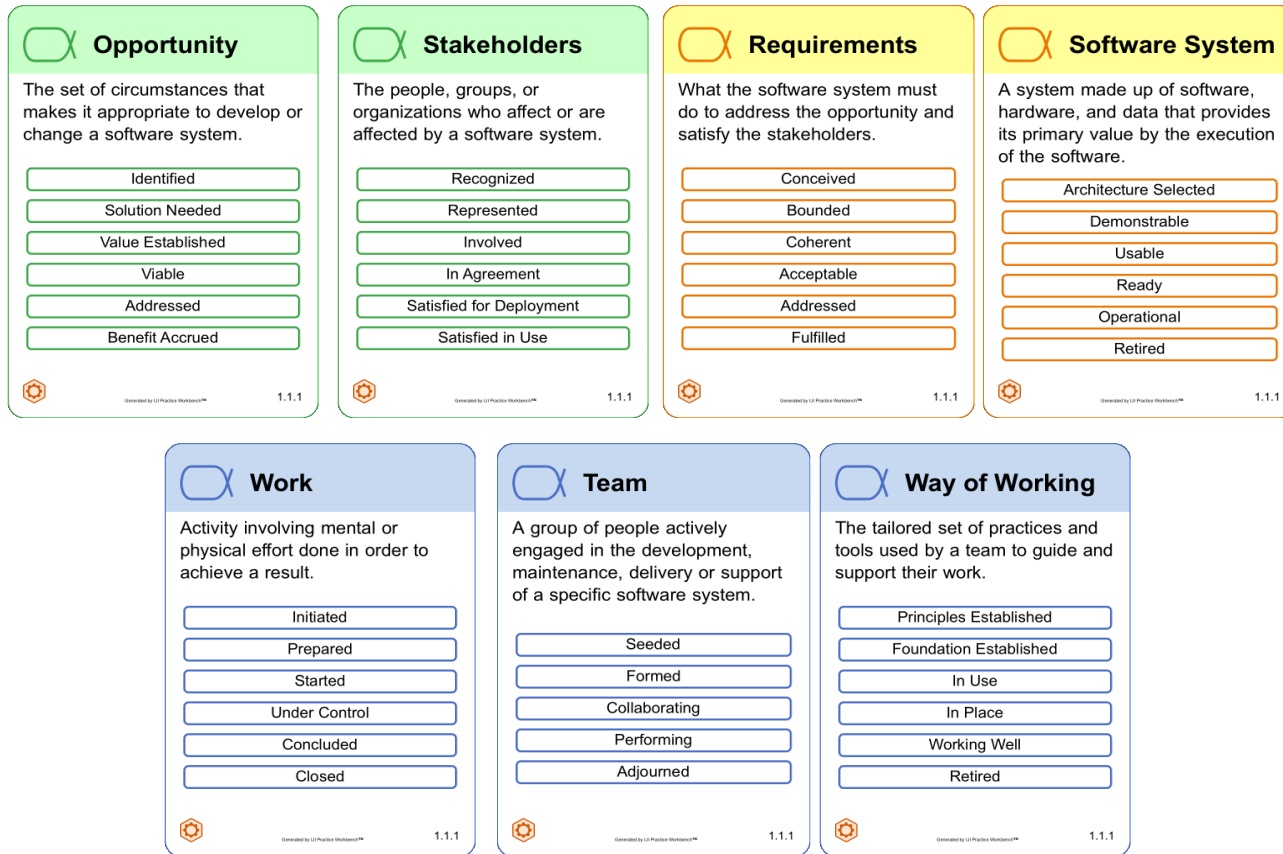
1. The alphas



We have already seen the Kernel Alphas



States of the Alphas in the Essence Kernel



- The OMG standard defines the states for each kernel alpha shown
- The details of each state can be found in the Essence standard, and we will not go deeper into each of them here
- You should be able to download them from the web site of the Essence book.



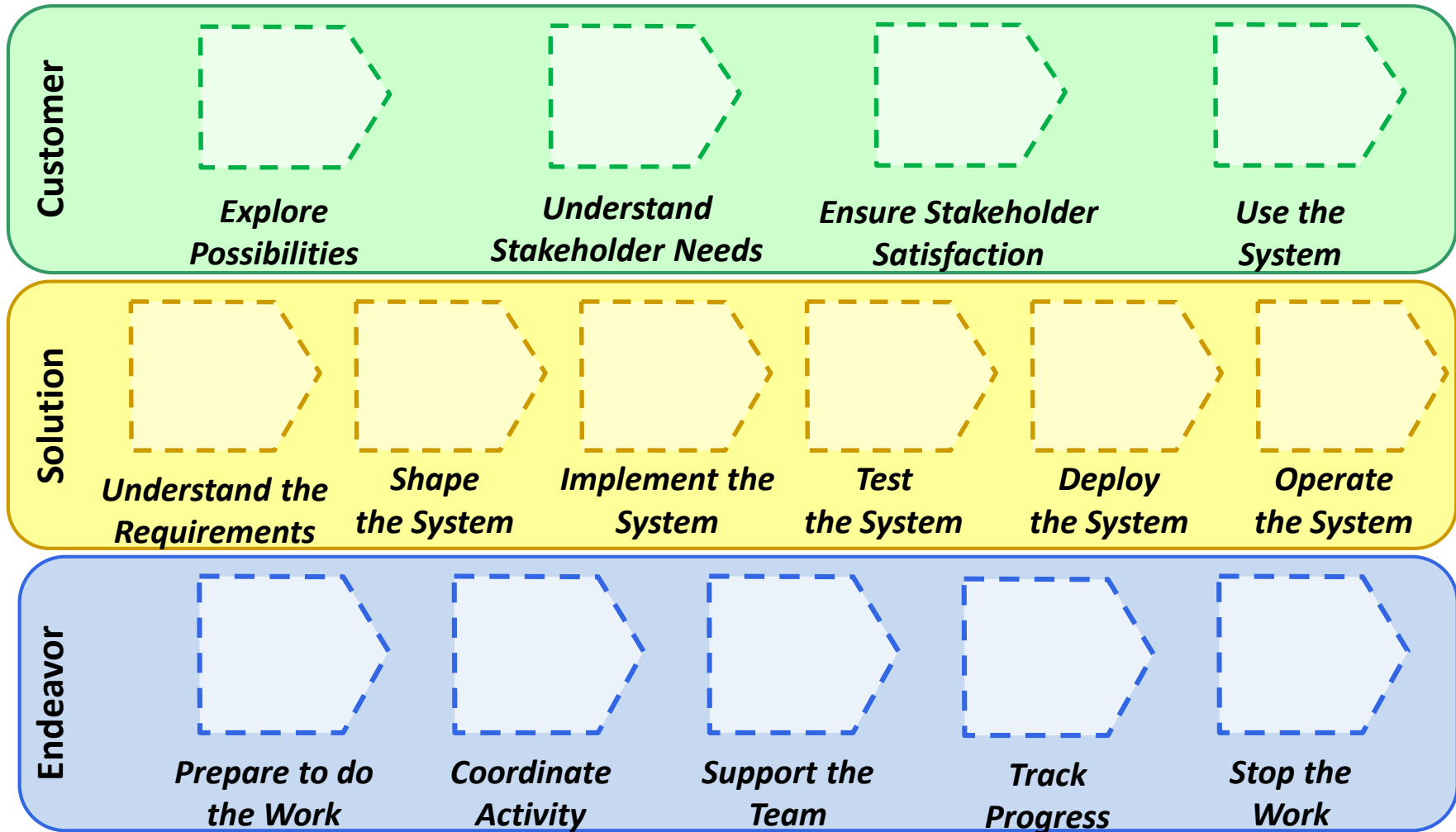
2. The Activities and Activity Spaces

- In every software development endeavour you carry out a number of activities.
 - **Essence does not define any activities**
 - how your team goes about capturing and communicating the requirements can be very different from team to team
 - **Essence defines a number of activity spaces.**
- **Def. Activity spaces are generic placeholders for specific activities**
 - Since the activity spaces are generic
 - They are method-independent
 - They could be standardized and are thus part of the Essence standard
 - Each activity space can be extended with concrete activities that progress one or more alphas
 - The activity spaces are packages used to group activities, which are related to one another
 - The activity spaces represent the essential things that have to be done to develop software



Activity Spaces in Kernel Standard

These are the Activity Space from Essence Standard



Activity Spaces Essence Standard Desc.

Customer

- **Explore Possibilities**
Explore the possibilities presented by the creation of a new or improved software system. This includes the analysis of the opportunity and the identification of the stakeholders.
- **Understand Stakeholder Needs**
Engage with the stakeholders to understand their needs and ensure that the right results are produced. This includes identifying and working with the stakeholder representatives to progress the opportunity.
- **Ensure Stakeholder Satisfaction**
Share the results of the development work with the stakeholders to gain their acceptance of the system produced and verify that the opportunity has been addressed.
- **Use the System**
Observe the use the system in a live environment and how it benefits the stakeholders.

Solution

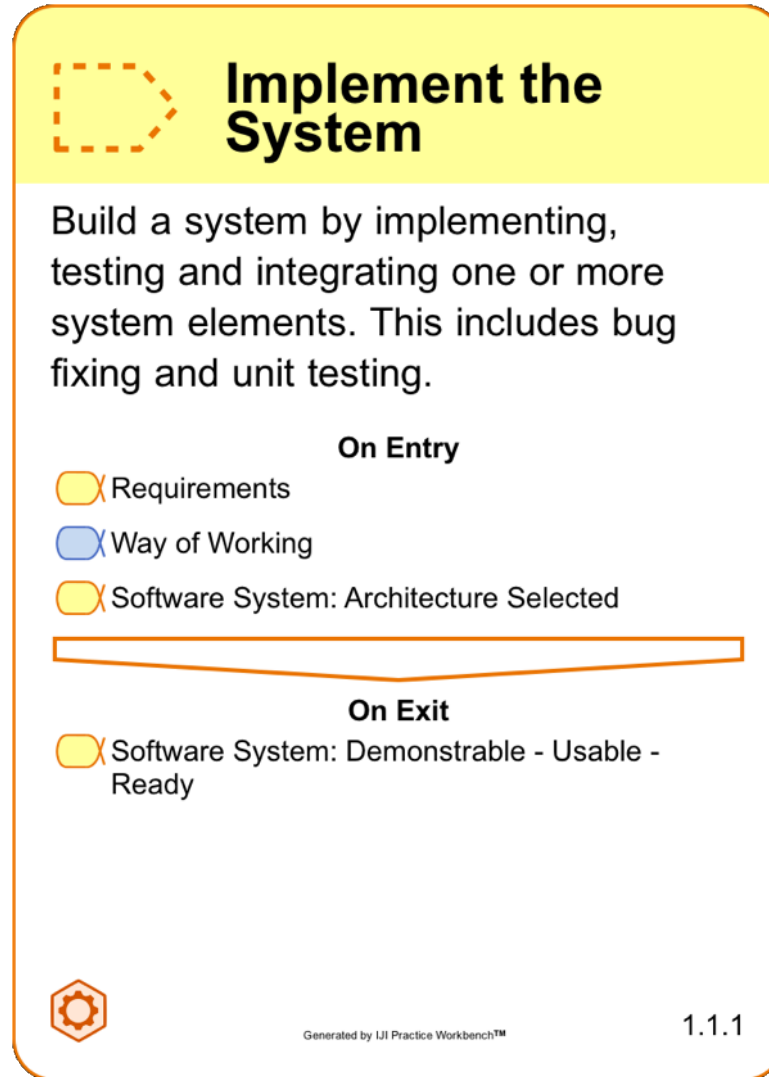
- **Understand the Requirements**
Establish a shared understanding of what the system to be produced must do.
- **Shape the system**
Shape the system so that it is easy to develop, change and maintain, and can cope with current and expected future demands. This includes the architecting and overall design of the system to be produced.
- **Implement the System**
Build a system by implementing, testing and integrating one or more system elements. This includes bug fixing and unit testing.
- **Test the System**
Verify that the system produced meets the stakeholders' requirements.
- **Deploy the System**
Take the tested system and make it available for use outside the development team

Endeavour

- **Prepare to do the Work**
Set up the team and its working environment. Understand and commit to completing the work.
- **Coordinate Activity**
Co-ordinate and direct the team's work. This includes all ongoing planning and re-planning of the work, and re-shaping of the team.
- **Support the Team**
Help the team members to help themselves, collaborate and improve their way of working.
- **Track Progress**
Measure and assess the progress made by the team.
- **Stop the Work**
Shut-down the software engineering endeavour and handover of the team's responsibilities.

Activity Space Card

- Activity space cards have very similar contents as activity cards



- Activity name
- Very brief Activity description
- Inputs for activity
- Outputs of activity

3. The Competencies



Def. *Competencies* are generic containers for specific skills

- Specific skills, for example Java programming, are not part of the kernel because this skill is not essential on all software engineering endeavours.
- But competency is always required and it will be up to the individual teams to identify the specific skills needed for their particular software endeavour.
- A common problem on software endeavours is not being aware of the gap between the competencies needed and the competencies available.
 - The kernel approach will raise the visibility of this gap.

Competences in Essence Kernel Standard

- Competencies are aligned to the three focus areas
- Essence Kernel Standard competencies are needed for any Software Engineering Endeavour, independently then methods and techniques adopted



Competences Essence Standard Desc.

Customer

- **Stakeholder Representation**

This competency encapsulates the ability to gather, communicate, and balance the needs of other stakeholders, and accurately represent their views.

Solution

- **Analysis**

This competency encapsulates the ability to understand opportunities and their related stakeholder needs, and to transform them into an agreed upon and consistent set of requirements.

- **Development**

This competency encapsulates the ability to design, program and code effective and efficient software systems following the standards and norms agreed upon by the team.

- **Testing**

This competency encapsulates the ability to test a system, verify that it is usable and that it meets the requirements.

Endeavour

- **Leadership**

This competency enables a person to inspire and motivate a group of people to achieve a successful conclusion to their work and to meet their objectives.

- **Management**

This competency encapsulates the ability to coordinate, plan and track the work done by a team

Competency levels

- Each of the competencies has a competency level
- The competency level is the same across all of the kernel competencies.

Competency levels of achievement:

- 1. Assists** – Demonstrates a basic understanding of the concepts and can follow instructions.
- 2. Applies** – Able to apply the concepts in simple contexts by routinely applying the experience gained so far.
- 3. Masters** – Able to apply the concepts in most contexts and has the experience to work without supervision.
- 4. Adapts** – Able to apply judgment on when and how to apply the concepts to more complex contexts. Can enable others to apply the concepts.
- 5. Innovates** – A recognized expert, able to extend the concepts to new contexts and inspire others.

4. Patterns



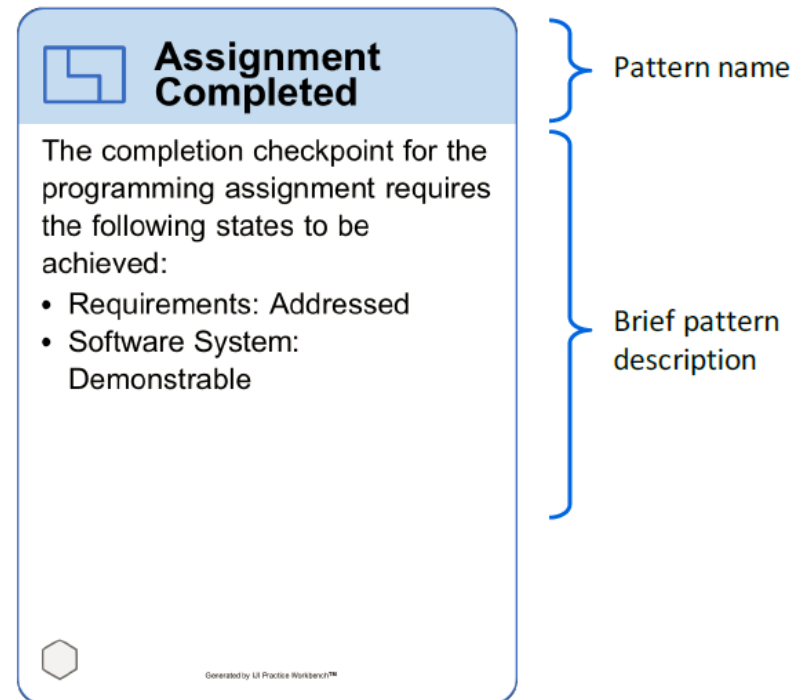
Def. *Patterns* are generic solutions to typical problems

- Patterns is the way *Essence* allows arrangements of elements to solve a specific problem
- Patterns are optional elements (not required element of a practice definition) that may be associated with any other language element.
- *Patterns* examples exist in our daily life as well as in Software Engineering:
 - In a classroom, we often see the teacher in front, with rows of desks and chairs for students. This is a common teaching pattern.
 - In SW Eng we use patterns very often. Some examples are:
 - CheckPoints, Student Pairs, etc.
- *Roles* are special type of *Patterns*



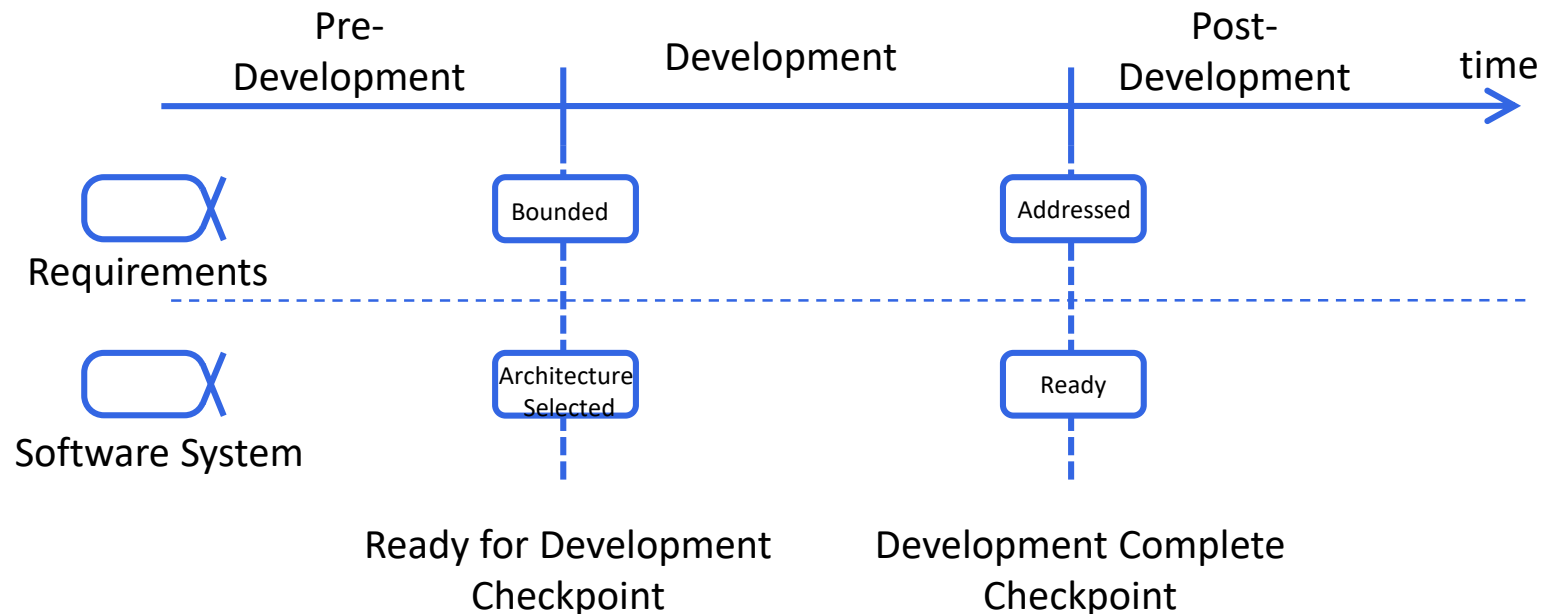
A Pattern Example: Checkpoint

- A *checkpoint* is a set of criteria to be achieved at a specific point in time where an important decision is to be taken.
 - A checkpoint is simply expressed by a set of alpha states that must have been achieved in order to pass the checkpoint.
- This pattern can be reused for other similar endeavours trying to get to the same checkpoint.



Using Checkpoint Pattern Example

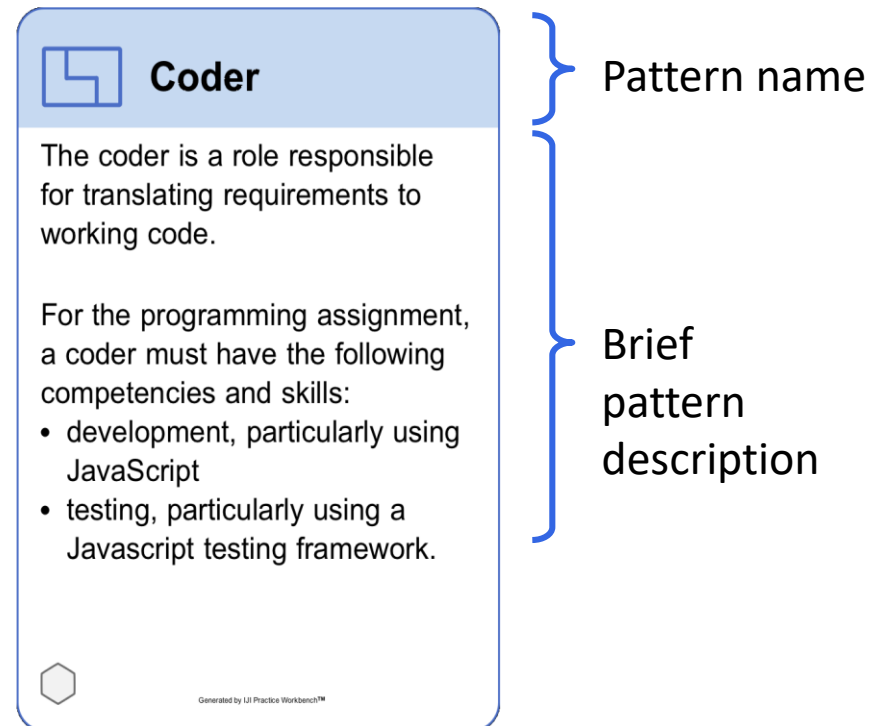
- Let's use Checkpoints to decide when to start and when to finish development of a software project



- In this example, there are two checkpoints.
 - What are the checkpoints?
- The criteria for these two checkpoints are expressed using alpha states.
 - What are the Alpha States for each Check Point?

Roles: A Special kind of Pattern

- Roles represent the set of competencies needed to do a job effectively
 - Roles are a special kind of pattern that apply to people
 - Example of Roles are Coder, Analyst, Tester
- Responsibilities to achieve a task are assigned to the task owner, that could be playing a role, but the responsibilities are not part of the role definition



Summary of Essence Elements and Cards



Alpha

Software System

A system made up of software, hardware, and data that provides its primary value by the execution of the software.

Architecture Selected

Demonstrable

Usable

Ready

Operational

Retired



Generated by U2 Practice Workbooks™

1.1.1

Work Product

Code

Good code that not only implements requirements, but also in a self-explanatory way.

Pseudo Coded

Code Completed

Code Explained

Describes:  Software System




Generated by U2 Practice Workbooks™

Activity

Write Code

Collaborate together to produce good quality code that meet requirements.


 Requirements: Bounded


Implement the System



Development

 Requirements: Addressed

 Software System: Ready

 Code: Code Completed



Generated by U2 Practice Workbooks™

Competency

Development

The ability to design and program effective software systems following the standards and norms agreed by the team.

Innovates 

Adapts 

Masters 

Applies 

Assists 



Generated by U2 Practice Workbooks™

Pattern

Assignment Completed

The completion checkpoint for the programming assignment requires the following states to be achieved:

- Requirements: Addressed
- Software System: Demonstrable



Generated by U2 Practice Workbooks™


Activity Area


Implement the System

Build a system by implementing, testing and integrating one or more system elements. This includes bug fixing and unit testing.


On Entry

 Requirements

 Way of Working

 Software System: Architecture Selected

On Exit

 Software System: Demonstrable - Usable - Ready



Generated by U2 Practice Workbooks™

1.1.1



**Essence Seminar
For University of Rome "Tor Vergata"**

Conclusions: *Reflections on Essence Goals and Theory of Software Engineering*

Giuseppe Calavaro, Ph.D.
IBM Big Data Practice Leader
External Professor at University of Rome "Tor Vergata"

www.semat.org

What is Essence?



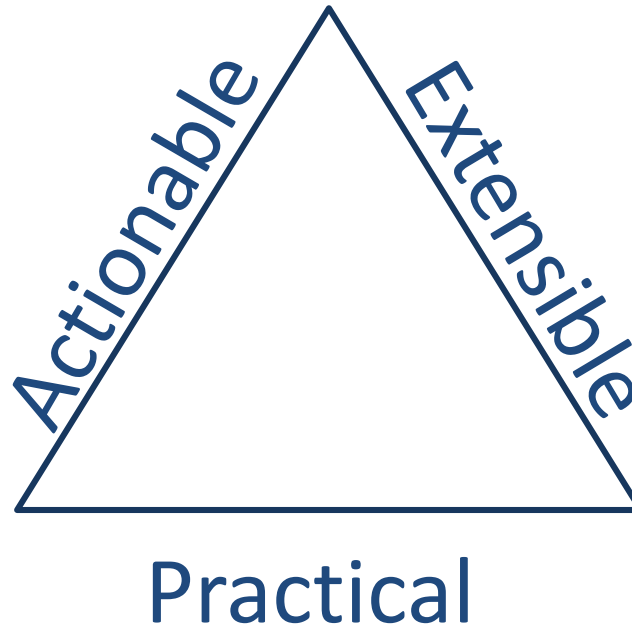
- Essence provides a common ground for Software Engineering
 - It is very important to have such common ground
 - It is more than a conceptual mode
 - It allows to represent any software engineering method
- Essence Kernel is
 - A **thinking framework** for teams to reason about the **progress** they are making and the **health** of their **endeavors**.
 - A **framework** for teams to **assemble** and continuously improve their **way of working**.
 - The **common ground for improved communication**, standardized measurement, and the sharing of best practice.
 - A **foundation for accessible, inter-operable method** and practice definitions.
 - And most importantly, a way to **help teams understand where they are, and what they should do** next.



What makes Essence more than a conceptual framework?

Essence Guiding principles

- Alphas helps assess & drive progress and health of project
- Each state has a checklist
 - Criteria needed to reach the state
- Alphas are method and practice independent



- Practices are distinct, separate, modular units
- Kernel allow create or tailor and compose practices to new methods
- Additional Alphas can be added

- Tangible through the cards
 - Cards provide concise reminders
- Practical through Checklists and Prompts
 - Utilizable on a daily basis helping making decisions

Essence and Agile (or other approaches)



- Essence Kernel doesn't compete with existing methods
- Essence kernel can be used with all the currently popular management and technical practices:
 - Scrum
 - Kanban
 - risk-driven
 - Iterative
 - Waterfall
 - use-case driven development
 - acceptance test driven development
 - continuous integration
 - test driven development
 - Etc.
- It will help all sizes of teams
 - from one-man bands to 1,000 strong software engineering programs.
- The kernel supports the values of the Agile Manifesto
 - It values the 'use of methods' over 'the description of method definitions'

What is a Theory?

- Most theories share three characteristics
 - they attempt to generalize local observations and data into more abstract and universal knowledge
 - they generally have an interest in causality (cause and effect)
 - they often aim to explain or predict a phenomenon.
- Gregor[REF] proposes 4 goals for a theory:
 1. Describe the studied phenomenon
 - Function Point and SWEBOK could serve as an example.
 2. Explain the how, why, and when of the topic
 - theory of cognition is aimed at explaining the human mind's limitations
 3. Beside explaining what has already happened also predict what will happen next
 - Cocomo attempts to predict the cost of software projects
 4. Prescribe how to act based on predictions
 - Alan Davis's 201 principles exemplify this goal[REF]

Where is the Theory for SW Engineering?

- Most academic disciplines are very concerned with their theories.
- Why the software engineering community seems so uninterested in discussing its theories?
- ***Software Engineering Doesn't Need Theory?***
 - Software engineering *isn't* doing fine.
 - All engineering fields need theory,
 - The maturity of scientific disciplines can be measured by the unity of their theories
- ***Software Engineering Already Has Its Theory?***
 - A discipline's significant theories should be able to provide answers to that discipline's significant questions...
- ***Software Engineering Can't Have a Theory?***
 - Software engineering is a practical engineering discipline without scientific ambitions where rules of thumb and guidelines assume the role of theory
 - We don't believe that there's any rational reason for the lack of theoretical focus in software engineering
 - Without the predictive and prescriptive support of theory, software engineering would be relegated to the horribly costly design process of trial and error

Essence is founding the theory of SW Eng



- Theory is generally used to
 1. describe a phenomenon of interest,
 2. to explain and predict that phenomenon
- Description precedes prediction and to describe something, a language is needed.
- There is currently no widely accepted predictive general theory of software engineering.
 - However, the Essence takes the first step by proposing a coherent, general, *descriptive* theory of software engineering, i.e. a language of software engineering.
 - But a complete consideration of the causality between concepts and thus prediction is beyond the current version of the Essence.

Conclusions



- Essence kernel is a spring board towards more mature software engineering practices and a more mature software engineering discipline.
- In the remaining parts of this course, we will demonstrate how Essence helps you and your teams collaborate more effectively.

Authors and affiliations

- This is the list of the main authors of the book “Essence of Software Engineering”. The ideas and positions are solely opinion of the authors and do not reflect their company positions.
- Ivar Jacobson
 - Ivar Jacobson International Founder and Chairman
- Harold “Bud” Lawson
 - Coordinating Editor Systems Series College Publications - Kings College, UK
- Pan-Wei Ng
 - Agile Lean Enterprise, ThoughtWorks & Nanyang Technological University, Singapore
- Paul McMahon
 - Principal Consultant at PEM Systems, USA
- Michael Goedicke
 - Vice Dean at University of Duisburg-Essen,
- Ian Spence
 - Ivar Jacobson International Chief Scientist & SAFe Fellow, UK