

# Essence and the Basics of Software Engineering



**Idaho State  
University**

Computer  
Science

Dr. Isaac Griffith

CS 3321  
Department of Computer Science  
Idaho State University

**ROAR**

# Outcomes

After today's lecture you will:

- Understand the basic concepts related to software processes
  - Modeling processes
  - Waterfall process
  - Iterative processes
- Understanding the basics of Essence
- Understand the basics of Software Engineering
- Understand the basics of the Essence Language



# Software Engineering and Essence

- **Software Engineering** is the science to help developing software systems
  - Small and simple projects have different needs than large and complex projects, therefore there is no single approach or technology
  - Development organizations tailor, to the organization culture and experience as well as the specific projects, their software development processes.
- Essence is a way to essentialize Software Engineering to allow teams to define and **guide** their own Software Development Processes

# Software Development Process

- A Software Development Process, or Software Method, describes WHO does WHAT, HOW and WHEN to perform the ACTIVITIES that are needed to achieve the goal:

## **Implement a Software System**



# Waterfall

- It practically says that you:
  - ➊ Need to understand and describe what to build (Requirements)
  - ➋ Study the problem and Design a solution that will fulfill the requirements (Analysis and Design)
  - ➌ Implement a solution (Implementation)
  - ➍ Test and Verify that what you have realized works correctly and fulfills the requirements and expectations (Verification)
- Isn't that a logical approach?

# Waterfall

- The waterfall methods helped to bring some discipline to software engineering
  - Yet many people tried to follow literally which caused serious problems
- It requires that
  - you don't start the Analysis and Design until you have not gathered all requirements, and
  - You don't start implementation until you have finalized the design
- While it appears logical
  - Believing you can specify all requirements upfront is just a myth, also, requirements are always changing.
  - When implementing the solution, design flaws become evident requiring to rework the design where you spent already a lot of time

In short, real projects experiences have shown that:

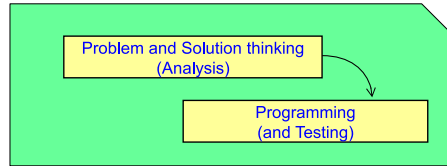
- **In real projects, and especially when implementing First of a Kind (FOAK) systems, this approach doesn't work**



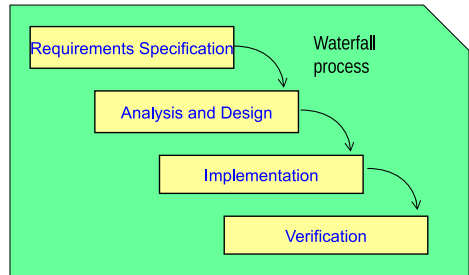
# Modeling Our Approach

- Initial Software Methods were very simple and could be represented by the models in these pictures.
- The Waterfall process is the most famous well defined process, defined to be used for large and complex software systems:
  - It is the most logical and easy to understand
  - It is very prescriptive and rigid
  - It doesn't lead to great results
    - Especially for projects on new first of a kind systems

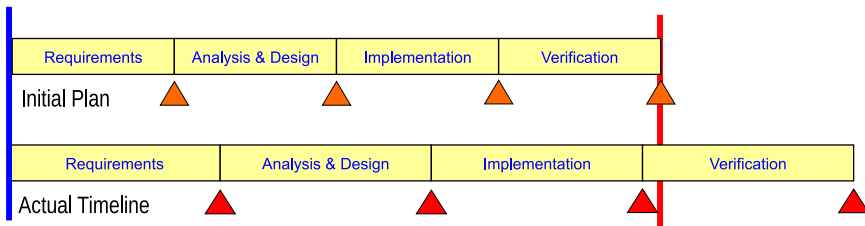
Steps to implement a small program for personal use



Steps of the initial waterfall process



# Waterfall Problem Examples



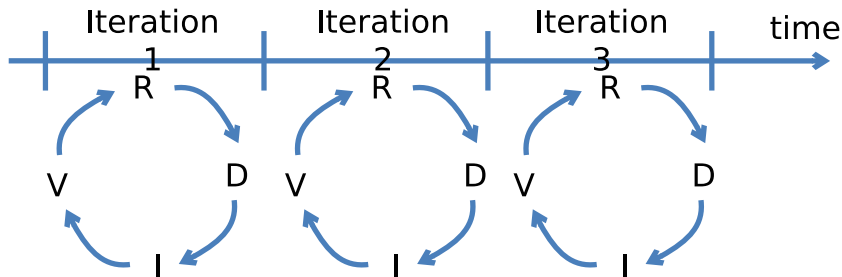
- As an explanation example:
  - let's assume we have a 4 months project made up of 4 phases of a month.  
Let's assume each one is late by one week. To deliver on time, we would need to skip practically the whole verification phase
- Project Delays are impacting future critical phases
- Risks are addressed late
- Testing is performed only at the end
- Project progress is measured by document releases
- It doesn't allow to release preliminary versions to evaluate stakeholder feedback



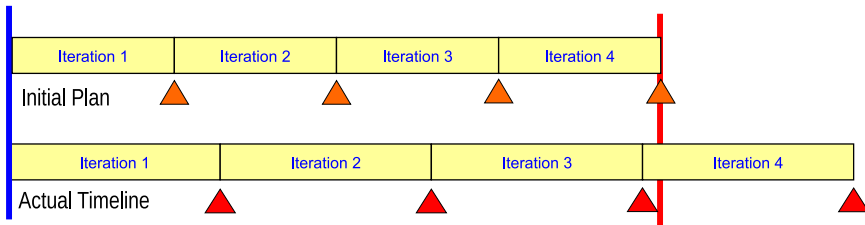


# The Iterative Approach

- Iterating means you can specify some requirements and you can build something meeting these requirements
  - As soon as you start to use what you have built, you will know how to make it a bit better.
  - Then you can specify some more requirements, and build and test these, until you have something that you feel can be released.



# Iterative Project Example



- As an explanation example:
  - let's assume we have a 4 months project made up of 4 iterations of a month. Let's assume each one is late by one week. We would deliver  $\frac{3}{4}$  of the system on time, well tested and stakeholders feedback considered.
- The Iteration 3 output is a system well tested
  - The effort and time planned for testing was spent for a quality work in each iteration
  - Stakeholder have had opportunities to provide feedback on early versions
  - Lower priority requirements will be in late iterations, creating low problem if missing when initial deployment is starting
- Usually time boxed iterations are used and is better to add new iterations rather than extending the duration of an iteration
- Risks are addressed in initial releases
- If required a smaller system can be released during the project.

# Methods, Techniques, and Practices

- Since the early days of Software Engineering, Methods wars are common:
    - Everyone described their own “infallible” method to develop software
  - With every major paradigm shift
    - such as the shift from function/data to object/component approaches and from the latter to the agile approaches
    - basically the industry throws out all they know about software development and start all over with new terminology with little relation to the old one.
  - Yet, in all these methods and techniques, there are useful Practices that can be very handy and useful in many cases
- Methods
    - SADT
    - DeMarco Method
    - Booch Method
    - OMT
    - OOSE
    - RUP/UP
    - XP
    - Agile Development
    - Etc.
  - Techniques
    - Object Oriented Programming
    - OO Design
    - Use Cases
    - User Stories
    - UML
    - Components
    - SOA, EA, PLA
    - Etc.



# Introduction to Essence

---

CS 3321

**ROAR**

# The Essence of our Goal

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

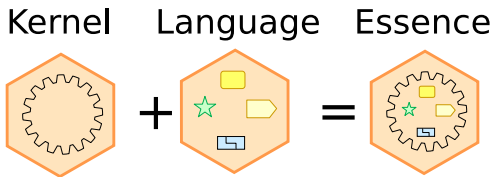
# Essence

- The Essence initiative come to light very recently to address the situation described above
- **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**
- As described in the Appendix, this theory is not yet fully shaped and Essence, though the first and most promising kernel of Software Engineering Theory, will need to evolve to become a fully fledged theory.

# 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 endeavors
- Language
  - The Essence language is very simple, intuitive and practical
  - Utilized in describing the Essence kernel with the elements that constitute a common ground



# Essentialized Practices and methods

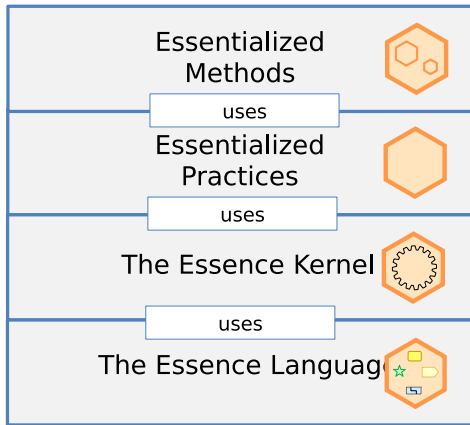
- Using Essence Kernel and Language you can Essentialize Practices and Methods
- Essentialize a practices means they are described using Essence
  - the Essence kernel
  - the Essence language
  - It focuses the description of the method/practice on what is essential.
- Consequently, the methods we describe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





# Essence Method Architecture

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



# Methods are Compositions of Practices

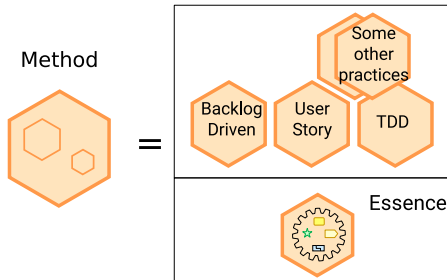
- Composition of practices is an operation merging two or more practices to form a method.
  - The operation has been defined mathematically in order to be precise.
  - The operation has to be specified by an expert with the objective to resolve potential overlaps and conflicts between the practices 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.

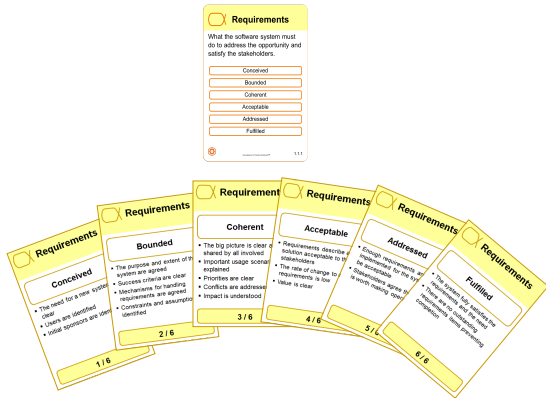
# How to Learn a Method

- 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 starts 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





# Basics of Software Engineering

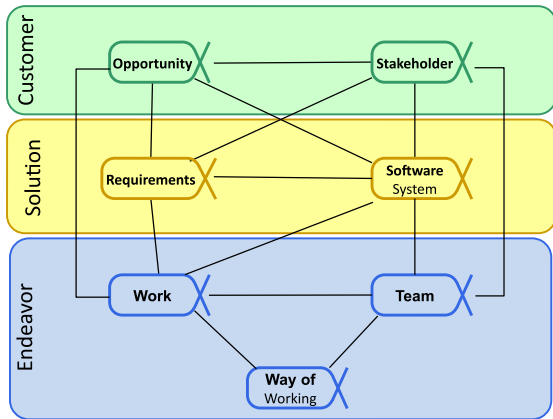
---

CS 3321

**ROAR**

# 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
- Endeavor**
  - 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



# Solution

## 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 endeavor 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 work 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



# The Essence Language

---





CS 3321

**ROAR**

# 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.

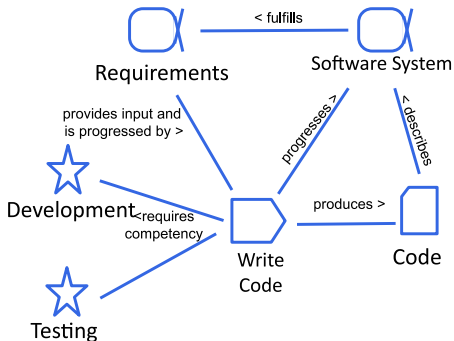
# Essence Language Element Types

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.

# 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.



# Alphas

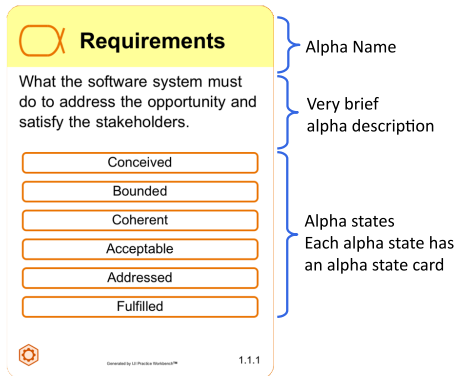
## Definition

**Alphas** are subjects in a software endeavor 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 endeavor.
- 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 Alphas
- 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

An Alpha Card provides a short and crisp description of an Alpha and it's states.





# Alpha State Cards

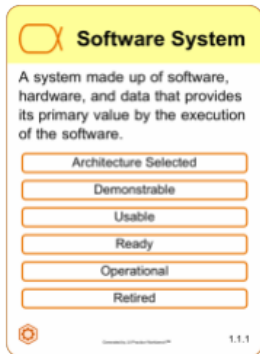
For each State of an Alpha, there is a card describing the checklist criteria to achieve

- 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 endeavor.
- 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	Requirements	Requirements
<p><b>Conceived</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Stakeholders agree system is to be produced</li> <li><input type="checkbox"/> Users identified</li> <li><input type="checkbox"/> Funding stakeholders identified</li> <li><input type="checkbox"/> Opportunity clear</li> </ul> <p>1 / 6</p> <p>1.1.1</p>	<p><b>Bounded</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Development stakeholders identified</li> <li><input type="checkbox"/> System purpose agreed</li> <li><input type="checkbox"/> System success clear</li> <li><input type="checkbox"/> Shared solution understanding exists</li> <li><input type="checkbox"/> Requirement's format agreed</li> <li><input type="checkbox"/> Requirements management in place</li> <li><input type="checkbox"/> Prioritization scheme clear</li> <li><input type="checkbox"/> Constraints identified &amp; considered</li> <li><input type="checkbox"/> Assumptions clear</li> </ul> <p>2 / 6</p> <p>1.1.1</p>	<p><b>Coherent</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Requirements shared</li> <li><input type="checkbox"/> Requirements' origin clear</li> <li><input type="checkbox"/> Rationale clear</li> <li><input type="checkbox"/> Conflicts addressed</li> <li><input type="checkbox"/> Essential characteristics clear</li> <li><input type="checkbox"/> Key usage scenarios explained</li> <li><input type="checkbox"/> Priorities clear</li> <li><input type="checkbox"/> Impact understood</li> <li><input type="checkbox"/> Team knows &amp; agrees on what to deliver</li> </ul> <p>3 / 6</p> <p>1.1.1</p>
<p><b>Acceptable</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Acceptable solution described</li> <li><input type="checkbox"/> Change under control</li> <li><input type="checkbox"/> Value to be realized clear</li> <li><input type="checkbox"/> Clear how opportunity addressed</li> <li><input type="checkbox"/> Testable</li> </ul> <p>4 / 6</p> <p>1.1.1</p>	<p><b>Addressed</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Enough addressed to be acceptable</li> <li><input type="checkbox"/> Requirements and system match</li> <li><input type="checkbox"/> Value realized clear</li> <li><input type="checkbox"/> System worth making operational</li> </ul> <p>5 / 6</p> <p>1.1.2</p>	<p><b>Fulfilled</b></p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Stakeholders accept requirements</li> <li><input type="checkbox"/> No hindering requirements</li> <li><input type="checkbox"/> Requirements fully satisfied</li> </ul> <p>6 / 6</p> <p>1.1.1</p>

# Software System Alpha Example

The programming Practice in our example also has 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.
- **Usable** – 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.

# Work Products

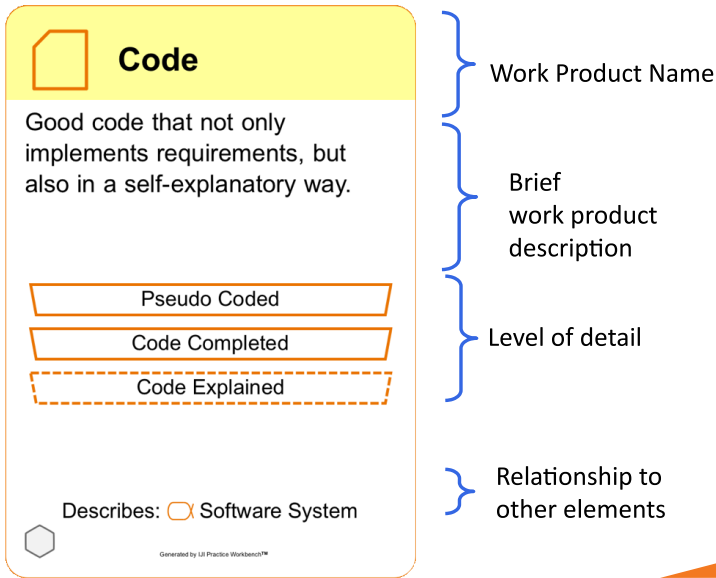
## Definition

***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



# Activity

## Definition

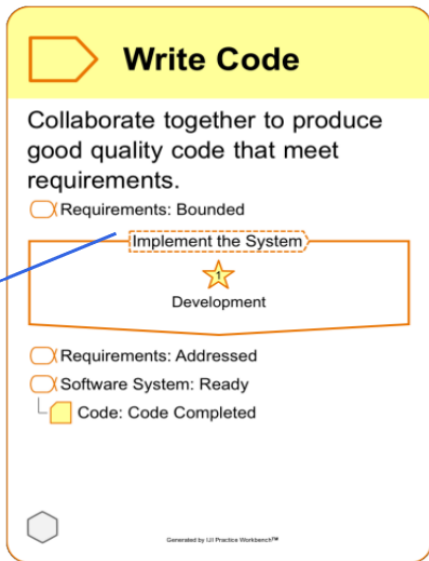
### **Activities are things which practitioners do**

- Activities examples are: holding a meeting, analyzing 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

## Definition

***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 endeavor.



# Competency Card



## Development

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

Innovates	5
Adapts	4
Masters	3
Applies	2
Assists	1



Generated by UI Practice Workbench™

Competency name

Brief  
Competency  
description

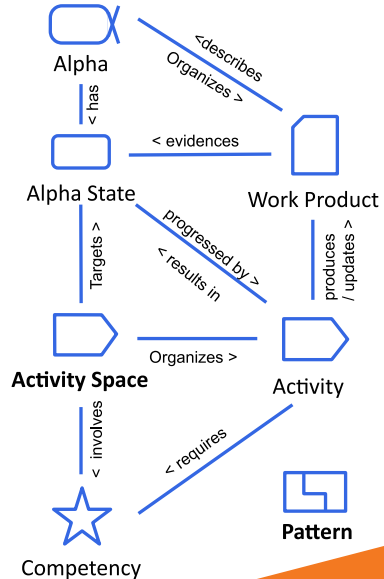
Competency levels







# Essence Language

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



# Additional Language Elements

- 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 in deeper detail in future lectures.

# Essentializing a Practices

- 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.

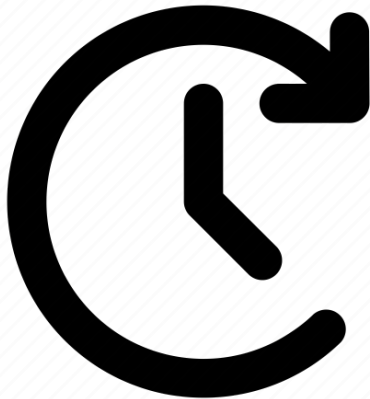
# Alphas vs. Products

Essence distinguishes between elements of health and progress vs. 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: **Work Products**
  - Work products are tangible things such as documents, which can have *different levels of detail*.

# For Next Time

- Review Essentials Chapters 2 - 5
- Review this Lecture
- Come to Class
- Read Essentials Chapter 6 and 7
- Review Lecture 03
- Watch Lecture 03





**Are there any questions?**