Essence – Kernel and Language for Software Engineering Methods

Revised Submission


OMG Document Number: ad/2012-11-01
Standard document URL: http://www.omg.org/spec/Essence/1.0
Associated File(s)*:
   OMG Document Number: ad/ 2012-11-02
   XMI: Essence.xmi
4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Activity
An activity defines one or more kinds of work items and gives guidance on how to perform these.

Activity space
A placeholder for something to be done in the software engineering endeavor. A placeholder may consist of zero to many activities.

Alpha
An essential element of the software engineering endeavor that is relevant to an assessment of the progress and health of the endeavor. Alpha is an acronym for an Abstract-Level Progress Health Attribute.

Alpha association
An alpha association defines a relationship between two alphas.

Area of concern
Elements in kernels or practices may be divided into a collection of main areas of concern that a software engineering endeavor has to pay special attention to. All elements fall into at most one of these.

Check list item
A check list item is an item in a check list that needs to be verified in a state.

Competency
A characteristic of a stakeholder or team member that reflects the ability to do work.
A competency describes a capability to do a certain job. A competency defines a sequence of competency levels ranging from a minimum level of competency to a maximum level. Typically, the levels range from 0 – assists to 5 – innovates. (See Section 8.1.6 and Section 9.3.4.)

Constraints
Restrictions, policies, or regulatory requirements the team must comply with.

Enactment
The act of applying a method for some particular purpose, typically an endeavor.

Endeavor
An activity or set of activities directed towards a goal.

Invariant
An invariant is a proposition about an instance of a language element which is true if the instance is used in a language construct as intended by the specification.

Kernel
A kernel is a set of elements used to form a common ground for describing a software engineering endeavor.
Method
A method is a composition of practices forming a (at the desired level of abstraction) description of how an endeavor is performed. A team’s method acts as a description of the team’s way-of-working and provides help and guidance to the team as they perform their task. The running of a development effort is expressed by a used method instance. This instance holds instances of alphas, work products, activities, and the like that are the outcome from the real work performed in the development effort. The used method instance includes a reference to the defined method instance, which is selected as the method to be followed.

Opportunity
The set of circumstances that makes it appropriate to develop or change a software system.

Pattern
A pattern is a description of a structure in a practice.

Practice
A repeatable approach to doing something with a specific purpose in mind.
A practice provides a systematic and verifiable way of addressing a particular aspect of the work at hand. It has a clear goal expressed in terms of the results its application will achieve. It provides guidance to not only help and guide practitioners in what is to be done to achieve the goal but also to ensure that the goal is understood and to verify that it has been achieved. (See Section 9.3.1.13.)

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

Role
A set of responsibilities.

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

Stakeholders
The people, groups, or organizations who affect or are affected by a software system.

State
A state expresses a situation where some condition holds.

State Graph
A state graph is a directed graph of states with transitions between these states. It has a start state and may have a collection of end states.

Team
The group of people actively engaged in the development, maintenance, delivery and support of a specific software system.

Transition
A transition is a directed connection from one state in a state machine to a state in that state machine.

Way-of-working
The tailored set of practices and tools used by a team to guide and support their work.
Work
Work is defined as all mental and physical activities performed by the team to produce a software system.

Work item
A piece of work that should be done to complete the work. It has a concrete result and it leads to either a state change or a confirmation of the current state. Work item may or may not have any related activity.

5 Symbols and Abbreviations

5.1 Symbols
There are no symbols defined in this specification.

5.2 Abbreviations
- Sub-alpha: Subordinate alpha

6 Additional Information

6.1 Submitting Organizations
The following organizations submitted this specification:
- Fujitsu/Fujitsu Services
- Ivar Jacobson International AB
- Model Driven Solutions
- SOFTEAM
- Universidad Nacional Autónoma de México (UNAM)

6.2 Supporting Organizations
The following organizations supported this specification:
- Alarcos Research Group, University of Castilla – La Mancha (UCLM)
- Florida Atlantic University
- General Direction of Computing and Information Technologies and Communication (DGTIC), National Autonomous University of Mexico (UNAM)
- Graduate Science and Engineering Computing, National Autonomous University of Mexico (UNAM)
- IICT-BAS
- Impetus
- InfoBLOCK
- International Business Machines Corporation
7 Overview of the Specification

7.1 Introduction to Essence

The work behind Essence is the Semat initiative\(^1\), \(^2\), \(^3\) – Software Engineering Method and Theory – that was incepted at the end of 2009. Semat addresses the many issues that challenge the field of software engineering. For example, the reliance on fads and fashions, the lack of a theoretical basis, the abundance of unique methods that are hard to compare, the dearth of experimental evaluation and validation, and the gap between academic research and its practical application in industry.

Successfully developing software systems benefit from the application of effective methods and well-defined processes, as indicated in the RFP. Traditionally, a method definition is thought of as being instantiated, and the activities – created from the definition – are executed by practitioners (e.g., analysts, developers, testers, project leads) in some predefined order to get the result, specified by the definition. These software method engineering approaches are often considered by development teams as being too heavyweight and inflexible. The view – “the team is the computer, the process is the program” – is not suitable for creative work like software engineering that requires support for work, which is agile, trial-and-error based and collaboration intensive.

Essence defines a Kernel and a Language for software engineering method specification. They are scalable, extensible, and easy to use, and allow people to describe the essentials of their existing and future methods and practices so that they can be compared, evaluated, tailored, used, adapted, simulated and measured by practitioners as well as taught and researched by academics and researchers. The Kernel provides the common ground to among other things help practitioners to compare methods and make better decisions about their practices. One of the most important features is that the Kernel elements form the basis of a vocabulary – a map of the software engineering context. The map would be used as a base on top of which we can define and describe any method or practice in existence or foreseen in the near future. The Kernel should also be extensible to care for new technologies, new practices, new social working patterns, and new research. This is also an application of the principle of separation of concerns: separating the kernel elements from the specifics of the different methods.

The kernel elements are always prevalent in any software endeavors. They are what we already have (e.g. teams and work), what we already do (e.g. specify and implement), and what we already produce (e.g. software systems) when we develop software. An important goal is that the Kernel is small and light at its base but extensible to cover more advanced uses, such as dealing with life-, safety-, business-, mission-, and security-critical systems.

The Kernel and its elements are defined using a domain-specific language (the domain being practices for software development), which has a static base (syntax and well-formedness rules) to allow defining methods effectively, and with additional dynamic features (operational semantics) to enable usage, and adaption. In addition, the language is also used to define practices and methods.

Practices are described using the Kernel elements; they also allow a practice to be merged with other relevant practices to form a higher-level “method” or composed practice. The elements in the Kernel must be defined in a way that allows them to be extensible and tailorable supporting a wide variety of practices, methods, and development teams. The key concepts include:

- A Method is a composition of practices. Methods are dynamic and used. Methods are not just descriptions for developers to read, they are dynamic, supporting their day-to-day activities. This changes the conventional definition of a method. A method is not just a description of what is expected to be done, but a description of what is actually done.

\(^1\) Software Engineering Method and Theory (Semat) website: [www.semat.org](http://www.semat.org)
A Practice is a repeatable approach to doing something with a specific purpose in mind. A practice provides a systematic and verifiable way of addressing a particular aspect of the work at hand.

- The Kernel includes essential elements of software engineering.
- The Language is the domain-specific language to define methods, practices and the essential elements of the kernel.

The relationships among these concepts are depicted in Figure 2.

The language design was driven by two main objectives: making methods visible to developers and making methods useful to developers. The first objective led to the definition of both textual and graphical syntax as well as to the development of a concept of views in the latter. This way, developers can represent methods in exactly the way that suits their purposes best. By providing both textual and graphical syntax, nobody is forced to use a graphical notation in situations where textual notation is easier to handle, and vice versa. By providing a concept of views, nobody is forced to show a complete graphical representation in situations where a partial graphical representation of a method is sufficient.

The second objective led to the definition of dynamic semantics for methods. This way, a method is more than a static definition of what to do, but an active guide for a team’s way-of-working. At any point in time in a running software engineering endeavor, a method can be consulted and it returns advice on what to do next. Moreover, a method can be tweaked at any point in time and still returns (a possibly alternate) advice on what to do next for the same situation.

### 7.2 The Key Differentiators

The Essence work is built on the experiences and lessons learnt in the software development community. Some of the key differentiators set this work apart from what has been done in the past. These are the following:

1. Finding the essence of software engineering and finding a way to embody that essence in a kernel enables us to build our knowledge on top of what we have known and learnt, and apply and reuse gained knowledge across different application domains and software systems of differing complexity.
2. Work with methods in an agile way that are as close to practitioners’ practice as possible, so that they can evolve the methods and adapt them to their particular context.
3. Apply the principle of Separation of Concerns (SoC) that puts focus on the things that matter the most.

---

5 Ivar Jacobson, Pan-Wei Ng, Paul E. McMahon, Ian Spence, Svante Lidman. The Essence of Software Engineering – Applying the Semat Kernel, in preparation to be published.
a. Focusing on what helps the least experienced developers over what helps the more experienced developers. This is motivated by the understanding that the majority of the development community is not interested in method descriptions but rather the use of the method.

b. Supporting practitioners over process engineers. This is motivated by the conviction that process engineers should work on what practitioners’ need, based on the real work they must do on their software endeavor.

c. Emphasizing intuitive and concrete graphical syntax over formal semantics. This does not mean that the semantics is not as important nor as necessary. However, the description should be provided in a language that can be easily understood by the vast developer community whose interests are to quickly understand and use the language, rather than caring about the beauty of the language design. Hence, Essence pays extreme attention to syntax.

d. Focusing on method use over method definition. Most previous similar efforts have paid interest to method definition, i.e., how to capture methods. These efforts have not focused on how to support the use of a method in software endeavors. As a result, the methods became “shelf-ware” that are not relevant to practitioners who actually develop the software. This Essence proposal focuses on the use of methods so that developers themselves can take control of their own way of working and allow the method to evolve as their endeavor progresses.

For detailed descriptions of the Kernel and the Language please refer to Section 8 Kernel Specification and Section 10 Language Specification.
8 Kernel Specification

This section presents the specification for the Software Engineering Kernel. It begins with an overview of the kernel as a whole and its organization into the three areas of concern. This is followed by a description of each area of concern and its contents.

8.1 Overview

8.1.1 What is the Kernel?

The Software Engineering Kernel is a stripped-down, light-weight set of definitions that captures the essence of effective, scalable software engineering in a practice independent way.

The focus of the kernel is to define a common basis for the definition of software development practices, one that allows them to be defined and applied independently. The practices can then be mixed and matched to create specific software engineering methods tailored to the specific needs of a specific software engineering community, project, team or organization. The kernel has many benefits including:

- It allows you to apply as few or as many practices as you like.
- It allows you to easily capture your current practices in a reusable and extendable way.
- It allows you to evaluate your current practices against a technique neutral control framework.
- It allows you to align and compare your on-going work and methods to a common, technique neutral framework, and then to complement it with any missing critical practices or process elements.
- It allows you to start with a minimal method adding practices as the endeavor progresses and when you need them.

8.1.2 What is in the Kernel?

The kernel is described using a small subset of the Kernel Language. It is organized into three areas of concern, each containing a small number of:

- **Alphas** – representations of the essential things to work with. The Alphas provide descriptions of the kind of things that a team will manage, produce, and use in the process of developing, maintaining and supporting good software. They also act as the anchor for any additional sub-alphas and work products required by the software engineering practices.

- **Activity Spaces** – representations of the essential things to do. The Activity Spaces provide descriptions of the challenges a team faces when developing, maintaining and supporting software systems, and the kinds of things that the team will do to meet them.

To maintain its practice independence the kernel does not include any instances of the other language elements such as work products or activities. These only make sense within the context of a specific practice.

The best way to get an overview of the kernel as a whole is to look at the full set of Alphas and Activity Spaces and how they are related.

8.1.3 Organizing the Kernel

The Kernel is organized into three discrete areas of concern, each focusing on a specific aspect of software engineering. As shown in Figure 2, these are:

- **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 to do the specification and development of the software system.
8.1.4 Alphas: The Things to Work With

The kernel Alphas 1) capture the key concepts involved in software engineering, 2) allow the progress and health of any software engineering endeavor to be tracked and assessed, and 3) provide the common ground for the definition of software engineering methods and practices. The Alphas, their relationships and their owning areas of concern are shown in Figure 3.

Throughout the diagrams in the body of the kernel specification, the three areas of concern are distinguished with different color codes where green stands for customer, yellow for solution, and blue for endeavor. The colors will facilitate the understanding and tracking of which area of concern owns which Alphas and Activity Spaces.
In the **customer** area of concern the team needs to understand the stakeholders and the opportunity to be addressed:

1. **Opportunity**: The set of circumstances that makes it appropriate to develop or change a software system.
   The opportunity articulates the reason for the creation of the new, or changed, software system. It represents the team’s shared understanding of the stakeholders’ needs, and helps shape the requirements for the new software system by providing justification for its development.

2. **Stakeholders**: The people, groups, or organizations who affect or are affected by a software system.
   The stakeholders provide the opportunity and are the source of the requirements and funding for the software system. They must be involved throughout the software engineering endeavor to support the team and ensure that an acceptable software system is produced.

In the **solution** area of concern the team needs to establish a shared understanding of the requirements, and implement, build, test, deploy and support a software system that fulfills them:

3. **Requirements**: What the software system must do to address the opportunity and satisfy the stakeholders.
   It is important to discover what is needed from the software system, share this understanding among the stakeholders and the team members, and use it to drive the development and testing of the new system.

4. **Software System**: A system made up of software, hardware, and data that provides its primary value by the execution of the software.
   The primary product of any software engineering endeavor, a software system can be part of a larger software, hardware or business solution.

In the **endeavor** area of concern the team and its way-of-working have to be formed, and the work has to be done:

5. **Work**: Activity involving mental or physical effort done in order to achieve a result.
   In the context of software engineering, work is everything that the team does to meet the goals of producing a software system matching the requirements, and addressing the opportunity, presented by the customer. The work is guided by the practices that make up the team’s way-of-working.

6. **Team**: The group of people actively engaged in the development, maintenance, delivery and support of a specific software system.
   The team plans and performs the work needed to update and change the software system.

7. **Way-of-Working**: The tailored set of practices and tools used by a team to guide and support their work.
   The team evolves their way of working alongside their understanding of their mission and their working environment. As their work proceeds they continually reflect on their way of working and adapt it as necessary to their current context.

### 8.1.5 Activity Spaces: The Things to Do

The kernel also provides a set of activity spaces that complement the Alphas to provide an activity based view of software engineering. The kernel activity spaces are shown in Figure 5.

In the **customer** area of concern the team has to understand the opportunity, and support and involve the stakeholders:

- **Explore Possibilities**: Explore the possibilities presented by the creation of a new or improved software system. This includes the analysis of the opportunity to be addressed 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 successfully addressed.

- **Use the System**: Use the system in a live environment to benefit the stakeholders.
In the solution area of concern the team has to develop an appropriate solution to exploit the opportunity and satisfy the stakeholders:

- **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 overall design and architecting 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.
- **Operate the System**: Support the use of the software system in the live environment.

In the endeavor area of concern the team has to be formed and progress the work in-line with the agreed way-of-working:

- **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 on-going planning and re-planning of the work, and adding any additional resources needed to complete the formation 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 endeavor and the handover of the team’s responsibilities.
8.1.6 Competencies: The Abilities Needed

The kernel also provides a set of competencies that complement the Alphas and Activity Spaces to provide a view of the key competencies needed to do software engineering. The kernel competencies are shown in Figure 6.

![Figure 6 – The Kernel Competencies](image)

In the **customer** area of concern the team has to be able to demonstrate a clear understanding of the business and technical aspects of their chosen domain and have the ability to accurately reflect the views of their stakeholders. This requires the following competencies to be available to the team:

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

In the **solution** area of concern the team has to be able to capture and analyze the requirements, and build and operate a software system that fulfils them. This requires the following competencies to be available to the team:

- **Analysis**: This competency encapsulates the ability to understand opportunities and their related stakeholder needs, and transform them into an agreed and consistent set of requirements.
- **Development**: This competency encapsulates the ability to design and program effective software systems following the standards and norms agreed by the team.
- **Testing**: This competency encapsulates the ability to test a system, verifying that it is usable and that it meets the requirements.

In the **endeavor** area of concern the team has to be able to organize itself and manage its work load. This requires the following competencies to be available to the team:

- **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.

Each competency has five levels of achievement. These are standard across all of the kernel competencies and summarized in Table 7. The table reads from top to bottom with the lowest level of competency shown in the first row and the highest in the last row.
<table>
<thead>
<tr>
<th>Competency Level</th>
<th>Brief Description</th>
</tr>
</thead>
</table>
| 1 - Assists      | Demonstrates a basic understanding of the concepts and can follow instructions. The following describe the traits of a Level 1 individual:  
• Understands and conducts his or her self in a professional manner.  
• Is able to correctly respond to basic questions within his or her domain.  
• Is able to perform most basic functions within the domain.  
• Can follow instructions and complete basic tasks. |
| 2 - Applies      | Able to apply the concepts in simple contexts by routinely applying the experience gained so far. The following describe the traits of a Level 2 individual:  
• Is able to collaborate with others within the Team  
• Is able to satisfy routine demands and simple work requirements.  
• Can handle simple challenges with confidence.  
• Can handle simple work requirements but needs help in handling any complications or difficulties.  
• Is able to reason about the context and draw sensible conclusions. |
| 3 - Masters      | Able to apply the concepts in most contexts and has the experience to work without supervision. The following describe the traits of a Level 3 individual:  
• Is able to satisfy most demands and work requirements.  
• Is able to speak the domain language with ease and accuracy.  
• Is able to communicate and explain his or her work  
• Is able to give and receive constructive feedback  
• Knows the limits of his or her capability and when to call on more expert advice.  
• Works at a professional level with little or no guidance. |
| 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. The following describe the traits of a Level 4 individual:  
• Is able to satisfy complex demands and work requirements.  
• Is able to communicate with others working outside the domain.  
• Can direct and help others working within the domain.  
• Is able to adapt his or her way-of-working to work well with others, both inside and outside their domain. |
| 5 - Innovates    | A recognized expert, able to extend the concepts to new contexts and inspire others. The following describe the traits of a Level 5 individual:  
• Has many years of experience and is currently up to date in what is happening |
within the domain.

- Is recognized as an expert by his or her peers.
- Supports others in working on a complex professional level.
- Knows when to innovate or do something different and when to follow normal procedure.
- Develops innovative and effective solutions to the current challenges within the domain.

The higher competency levels build upon the lower ones. An individual at level 2 has all the traits of an individual at level 1 as well as the additional traits required to qualify for level 2. An individual at level 3 has all the traits required at levels 1, 2 and 3, and so on.

Individuals at levels 1 and 2 have an awareness or basic understanding of the knowledge, skills, and abilities associated with the competency. However, they do not possess the knowledge, skills, and abilities to perform the competency in difficult or complex situations and typically can only perform simple routine tasks without direction or other guidance.

Individuals at level 3 and above have mastered this aspect of their profession and can be trusted to integrate into, and deliver the results required by, the team.

There are many factors that drive up the level of competency required by a team, including:

- The size and complexity of the work.
- The size and distribution of the team.
- The size, complexity and diversity of the stakeholder community.
- The novelty of the solution being produced.
- The technical complexity of the solution.
- The levels of risk facing the team.

8.2 The Customer Area of Concern

8.2.1 Introduction

This area of concern contains everything to do with the actual use and exploitation of the software system to be produced. Software engineering always involves at least one customer for the software that it produces. The customer perspective must be integrated into the day-to-day work of the team to prevent an inappropriate solution from being produced.

8.2.2 Alphas

The customer area of concern contains the following Alphas:

- Stakeholders
- Opportunity

8.2.2.1 Stakeholders

Description

**Stakeholders:** The people, groups, or organizations who affect or are affected by a software system.

The stakeholders provide the opportunity, and are the source of the requirements for the software system. They are involved throughout the software engineering endeavor to support the team and ensure that an acceptable software
system is produced.

**States**

- **Recognized** Stakeholders have been identified.
- **Represented** The mechanisms for involving the stakeholders are agreed and the stakeholder representatives have been appointed.
- **Involved** The stakeholder representatives are actively involved in the work and fulfilling their responsibilities.
- **In Agreement** The stakeholder representatives are in agreement.
- **Satisfied for Deployment** The minimal expectations of the stakeholder representatives have been achieved.
- **Satisfied in Use** The system has met or exceeds the minimal stakeholder expectations.

**Associations**

- **provide**: Opportunity Stakeholders provide Opportunity.
- **support**: Team Stakeholders support Team.
- **demand**: Requirements Stakeholders demand Requirements.
- **use and consume**: Software System Stakeholders use and consume Software System.

**Justification: Why Stakeholders?**

Stakeholders are critical to the success of the software system and the work done to produce it. Their input and feedback help shape the software engineering endeavor and the resulting software system.

**Progressing the Stakeholders**

During the development of a software system the stakeholders progress through several state changes. As shown in Figure 5, they are **recognized, represented, involved, in agreement, satisfied for deployment and satisfied in use**. These states focus on the involvement and satisfaction of the stakeholders, from their recognition as stakeholders through their representation in the development activities to their satisfaction with the use of the resulting software system. They communicate the progression of the relationship with the stakeholders who are either directly involved in the software engineering endeavor or support it by providing input and feedback.
Figure 7 – The states of the Stakeholders

As indicated in Figure 7, the first thing to do is to make sure that the stakeholders affected by the proposed software system are recognized. This means that all the different groups of stakeholders that are, or will be, affected by the development and operation of the software system are identified.

The number and type of stakeholder groups to be identified can vary considerably from one system to another. For example, the nature and complexity of the system and its target operating environment, and the nature and complexity of the development organization will both affect the number of stakeholder groups affected by the system.

It is not always possible to have all the stakeholder groups involved. Focus should be primarily on the ones that are critical to the success of the software engineering endeavor. It is these stakeholder groups that need to be directly involved in the work. Their selection depends on the level of impact they have on the success of the software system and the level of impact the software system has on them. The stakeholder groups that assure quality, fund, use, support and maintain the software system should always be identified.

It is not enough to determine which stakeholder groups need to be involved, they will also need to be actively represented. This means that there will be one or more stakeholder representatives selected to represent each stakeholder group, or in some cases one stakeholder representative selected to represent all stakeholder groups, and help the team. To make the contribution of the stakeholder representatives as effective as possible, they must know their roles and responsibilities within the software engineering endeavor. Without defining clear roles and responsibilities, the software engineering endeavor runs the risk that some of its important aspects may get unintentionally omitted or neglected.

Once the stakeholder representatives have been appointed, the represented state is achieved. Here, the stakeholder representatives take on their agreed to responsibilities and feel fully committed to helping the new software system to succeed. Acting as intermediaries between their respective stakeholder groups and the team, they are now granted authority to carry out their responsibilities on behalf of their respective stakeholder groups.

The team needs to make sure that the stakeholder representatives are actively involved in the development of the software system. Here, the stakeholder representatives assist in the software engineering endeavor in accordance with their responsibilities. They provide feedback and take part in decision making in a timely manner. In cases when changes need to be done to the software system, or when the stakeholder group they represent suggests changes, the stakeholder representatives make sure that the changes are relevant and promptly communicated to the team. No software engineering endeavor is fixed from the beginning. Its requirements are continuously evolving as the opportunity changes.
or new limitations are identified. This requires the stakeholder representatives to be actively involved throughout the development and to be responsive to all the changes affecting their stakeholder group.

It may not always be possible to meet all the expectations of all the stakeholders. Hence, compromises will have to be made. In the in agreement state the stakeholder representatives have identified and agreed upon a minimal set of expectations which have to be met before the system is deployed. These expectations will be reflected in the requirements agreed by the stakeholder representatives.

Throughout the development the stakeholder representatives provide feedback on the system’s state from the perspective of their stakeholder groups. Once the minimal expectations of the stakeholder representatives have been achieved by the new software system they will confirm that it is ready for operational use and the satisfied for deployment state is achieved.

Finally, the stakeholders start to use the operational system and provide feedback on whether or not they are truly satisfied with what has been delivered. Achieving the satisfied in use state indicates that the new system has been successfully deployed and is delivering the expected benefits for all the stakeholder groups.

Understanding the current state of the stakeholders and how they are progressing towards being satisfied with the new system is a critical part of any software engineering endeavor.

Checking the progress of the Stakeholders

To help assess the state and progress of the stakeholders, the following checklists are provided:

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
</table>
| Recognized       | All the different groups of stakeholders that are, or will be, affected by the development and operation of the software system are identified.  
                   | There is agreement on the stakeholder groups to be represented. At a minimum, the stakeholders groups that fund, use, support, and maintain the system have been considered.  
                   | The responsibilities of the stakeholder representatives have been defined.                                                                                                                                                                                                                                                                  |
| Represented      | The stakeholder representatives have agreed to take on their responsibilities.  
                   | The stakeholder representatives are authorized to carry out their responsibilities.  
                   | The collaboration approach among the stakeholder representatives has been agreed.  
                   | The stakeholder representatives support and respect the team's way of working.                                                                                                                                                                                                                                                         |
| Involved         | The stakeholder representatives assist the team in accordance with their responsibilities.  
                   | The stakeholder representatives provide feedback and take part in decision making in a timely manner.  
                   | The stakeholder representatives promptly communicate changes that are relevant for their stakeholder groups.                                                                                                                                                                                                                       |
| In Agreement     | The stakeholder representatives have agreed upon their minimal expectations for the next deployment of the new system.  
                   | The stakeholder representatives are happy with their involvement in the work.  
                   | The stakeholder representatives agree that their input is valued by the team and treated with respect.  
                   | The team members agree that their input is valued by the stakeholder representatives and treated with respect.  
                   | The stakeholder representatives agree with how their different priorities and perspectives are
being balanced to provide a clear direction for the team.

| Satisfied for Deployment | The stakeholder representatives provide feedback on the system from their stakeholder group perspective.  
The stakeholder representatives confirm that the system is ready for deployment. |
|--------------------------|-------------------------------------------------------------------------------------------------|
| Satisfied in Use         | Stakeholders are using the new system and providing feedback on their experiences.  
The stakeholders confirm that the new system meets their expectations. |

## 8.2.2.2 Opportunity

### Description

**Opportunity**: The set of circumstances that makes it appropriate to develop or change a software system.

The opportunity articulates the reason for the creation of the new, or changed, software system. It represents the team’s shared understanding of the stakeholders’ needs, and helps shape the requirements for the new software system by providing justification for its development.

### States

<table>
<thead>
<tr>
<th>Identified</th>
<th>A commercial, social or business opportunity has been identified that could be addressed by a software-based solution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Needed</td>
<td>The need for a software-based solution has been confirmed.</td>
</tr>
<tr>
<td>Value Established</td>
<td>The value of a successful solution has been established.</td>
</tr>
<tr>
<td>Viable</td>
<td>It is agreed that a solution can be produced quickly and cheaply enough to successfully address the opportunity.</td>
</tr>
<tr>
<td>Addressed</td>
<td>A solution has been produced that demonstrably addresses the opportunity.</td>
</tr>
<tr>
<td>Benefit Accrued</td>
<td>The operational use or sale of the solution is creating tangible benefits.</td>
</tr>
</tbody>
</table>

### Associations

| focuses: Requirements | Opportunity focuses Requirements. |

### Justification: Why Opportunity?

Most software engineering work is initiated by the stakeholders that own and use the software system. Their inspiration is usually some combination of problems, suggestions and directives, which taken together provide the development team with an opportunity to create a new or improved software system. Occasionally it is the development team itself that originates the opportunity that they must then sell to the other stakeholders to get funding and support. In many cases the software system only provides part of the solution needed to exploit the opportunity and the development team must coordinate their work with other teams to ensure that they actually deliver a useful, and deployable system.

In all cases understanding the opportunity is an essential part of software engineering, as it enables the team to:

- Identify and motivate their stakeholders.
- Understand the value that the software system offers to the stakeholders.
- Understand why the software system is being developed.
- Understand how the success of the deployment of the software system will be judged.
- Ensure that the software system effectively addresses the needs of all the stakeholders.

It is the opportunity that unites the stakeholders and provides the motivation for producing a new or updated software system. It is by understanding the opportunity that you can identify the value, and the desired outcome that the stakeholders hope to realize from the use of the software system either alone or as part of a broader business, or technical solution.
**Progressing the Opportunity**

During the development of a software system the opportunity progresses through several state changes. As presented in Figure 8, these are identified, solution needed, value established, viable, addressed, and benefit accrued. These states indicate significant points in the team’s progression of the opportunity from the initial formulation of an idea to use a software system through to the accrual of benefit from its use. They indicate (1) when the opportunity is first identified, (2) when the opportunity has been analyzed and it has been confirmed that a solution is needed, (3) when the opportunity’s value is established and the desired outcomes required of the solution are clear, (4) when enough is known about the cost of creating and using the proposed solution that it is clear that the pursuit of the opportunity is viable, (5) when a solution is available that demonstrably shows that the opportunity has been addressed, and finally (6) when benefit has been accrued from the use of the resulting solution.

As shown in Figure 8, the opportunity is first identified. The opportunity could be to entertain somebody, learn something, make some money, or even to change the world. Regardless of the kind of opportunity presented, if it is not understood by the team it is unlikely that they will produce an appropriate software system. For software engineering endeavors the opportunity is usually identified by the stakeholders that own and use the software system, and typically takes the form of an idea for a way to improve the current way of doing something, increase market share or apply a new or innovative technology.

Different stakeholders will see the opportunity in different ways, and they will be looking for different results from any software system produced to address it. It is important that the different stakeholder perspectives are understood and used to increase the team’s understanding of the opportunity. Analyzing the opportunity to understand the stakeholder’s needs and any underlying problems is essential to ensure that an appropriate system is produced and a satisfactory return-on-investment is generated.

Once the opportunity has been analyzed, and it has been agreed that a software-based solution is needed, it is possible to determine the value that the solution is expected to generate. Progressing the opportunity to value established is an important step in determining whether or not to proceed with work to address the opportunity as it means that the prize is...
clear to everyone involved.

The next step is to establish the viability of the opportunity. An opportunity is viable when a solution can be envisaged that it is feasible to develop and deploy within acceptable time and cost constraints. Although addressing the opportunity may be a very valuable thing to do it is probably not a good idea if the resources expended will be greater than the benefits accrued.

Once it has been agreed that the opportunity is viable then the team can be confident that a software system can be produced that will not just address the opportunity but will be acceptable to all of the stakeholders. As releases of the software system become available their viability must be continuously checked to ensure that they meet the needs of the stakeholders. After a suitable software system has been made available then, as far as the development team is concerned, the opportunity has been addressed. It is now up to the users of the system to actually use it to generate value and make sure that for this opportunity there is benefit accrued.

It is important that the team understands the current state of the opportunity so that they can ensure that an appropriate software system is developed, one that will satisfy the stakeholders and result in a tangible benefit being accrued.

**Checking the Progress of the Opportunity**

To help assess the state of the opportunity and the progress being made towards its successful exploitation, the following checklists are provided:

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identified</td>
<td>An idea for a way of improving current ways of working, increasing market share or applying a new or innovative software system has been identified.</td>
</tr>
<tr>
<td></td>
<td>At least one of the stakeholders wishes to make an investment in better understanding the opportunity and the value associated with addressing it.</td>
</tr>
<tr>
<td></td>
<td>The other stakeholders who share the opportunity have been identified.</td>
</tr>
<tr>
<td>Solution Needed</td>
<td>The stakeholders in the opportunity and the proposed solution have been identified.</td>
</tr>
<tr>
<td></td>
<td>The stakeholders' needs that generate the opportunity have been established.</td>
</tr>
<tr>
<td></td>
<td>Any underlying problems and their root causes have been identified.</td>
</tr>
<tr>
<td></td>
<td>It has been confirmed that a software-based solution is needed.</td>
</tr>
<tr>
<td></td>
<td>At least one software-based solution has been proposed.</td>
</tr>
<tr>
<td>Value Established</td>
<td>The value of addressing the opportunity has been quantified either in absolute terms or in returns or savings per time period (e.g. per annum).</td>
</tr>
<tr>
<td></td>
<td>The impact of the solution on the stakeholders is understood.</td>
</tr>
<tr>
<td></td>
<td>The value that the software system offers to the stakeholders that fund and use the software system is understood.</td>
</tr>
<tr>
<td></td>
<td>The success criteria by which the deployment of the software system is to be judged are clear.</td>
</tr>
<tr>
<td></td>
<td>The desired outcomes required of the solution are clear and quantified.</td>
</tr>
<tr>
<td>Viable</td>
<td>A solution has been outlined.</td>
</tr>
<tr>
<td></td>
<td>The indications are that the solution can be developed and deployed within constraints.</td>
</tr>
<tr>
<td></td>
<td>The risks associated with the solution are acceptable and manageable.</td>
</tr>
<tr>
<td></td>
<td>The indicative (ball-park) costs of the solution are less than the anticipated value of the</td>
</tr>
</tbody>
</table>
opportunity. The reasons for the development of a software-based solution are understood by all members of the team. It is clear that the pursuit of the opportunity is viable.

<table>
<thead>
<tr>
<th>Addressed</th>
<th>A usable system that demonstrably addresses the opportunity is available. The stakeholders agree that the available solution is worth deploying. The stakeholders are satisfied that the solution produced addresses the opportunity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benefit Accrued</td>
<td>The solution has started to accrue benefits for the stakeholders. The return-on-investment profile is at least as good as anticipated.</td>
</tr>
</tbody>
</table>

### 8.2.3 Activity Spaces

The customer area of concern contains four activity spaces that cover the discovery of the opportunity and the involvement of the stakeholders:

#### 8.2.3.1 Explore Possibilities

**Description**

Explore the possibilities presented by the creation of a new or improved software system. This includes the analysis of the opportunity to be addressed and the identification of the stakeholders.

Explore possibilities to:

- Enable the right stakeholders to be involved.
- Understand the stakeholders’ needs.
- Identify opportunities for the use of the software system.
- Understand why the software system is needed.
- Establish the value offered by the software system.

**Input:** None

**Completion Criteria:** Stakeholders::Recognized, Opportunity::Identified, Opportunity::Solution Needed, Opportunity::Value Established.

#### 8.2.3.2 Understand Stakeholder Needs

**Description**

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.

Understand stakeholder needs to:

- Ensure the right solution is created.
- Align expectations.
- Collect feedback and generate input.
- Ensure that the solution produced provides benefit to the stakeholders.

**Input:** Stakeholders, Opportunity, Requirements, Software System
8.2.3.3 Ensure Stakeholder Satisfaction

Description
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 successfully addressed.

Ensure the satisfaction of the stakeholders to:

- Get approval for the deployment of the system.
- Validate that the system is of benefit to the stakeholders.
- Validate that the system is acceptable to the stakeholders.
- Independently verify that the system delivered is the one required.
- Confirm the expected benefit that the system will provide.

**Input:** Stakeholders, Opportunity, Requirements, Software System

**Completion Criteria:** Stakeholders::Satisfied for Deployment, Opportunity::Addressed

8.2.3.4 Use the System

Description
Use the system in a live environment to benefit the stakeholders.

Use the system to:

- Generate measurable benefits.
- Gather feedback from the use of the system.
- Confirm that the system meets the expectations of the stakeholders.
- Establish the return-on-investment for the system.

**Input:** Stakeholders, Opportunity, Requirements, Software System

**Completion Criteria:** Stakeholders::Satisfied in Use, Opportunity::Benefit Accrued

8.2.4 Competencies

8.2.4.1 Stakeholder Representation

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

The stakeholder representation competency is the empathic ability to stand in for and accurately reflect the opinions, rights and obligations of other stakeholders.

People with this competency help the team to:

- Understand the business opportunity
- Understand the complexity and needs of the customers, users and other stakeholders
- Negotiate and prioritize the requirements
- Interact with the stakeholders and developers about the solution to be developed
- Understand how well the system produced addresses the stakeholders’ needs
Essential skills include:

- Negotiation
- Facilitation
- Networking
- Good written and verbal communication skills
- Empathy

This competency can be provided by an on-site customer, a product manager or a group of people from the business organization.

**Competency Levels**

Level 1 – Assists Demonstrates a basic understanding of the concepts and can follow instructions.

Level 2 – Applies Able to apply the concepts in simple contexts by routinely applying the experience gained so far.

Level 3 – Masters Able to apply the concepts in most contexts and has the experience to work without supervision.

Level 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.

Level 5 – Innovates A recognized expert, able to extend the concepts to new contexts and inspire others.

**Justification: Why Stakeholder Representation?**

When developing software it is essential to interact with the stakeholder community. However, it is impossible to directly interact with all of the stakeholders all of the time. This leads to a small number of stakeholders being selected to represent their particular stakeholder communities. For the smooth running of the team it is essential that the people selected have the competency needed to represent their stakeholder communities. The stakeholder representation competency encapsulates the abilities needed to be able to represent and act on behalf of others within a software engineering endeavor.

**8.3 The Solution Area of Concern**

**8.3.1 Introduction**

This area of concern covers everything to do with the specification and development of the software system.

The goal of software engineering is to develop working software as part of the solution to some problem. Any method adopted must describe a set of practices to help the team produce good quality software in a productive and collaborative fashion.

**8.3.2 Alphas**

The solution area of concern contains the following Alphas:

- Requirements
- Software System

**8.3.2.1 Requirements**

**Description**

Requirements: What the software system must do to address the opportunity and satisfy the stakeholders.
It is important to discover what is needed from the software system, share this understanding among the stakeholders and the team members, and use it to drive the development and testing of the new system.

**States**

- Conceived: The need for a new system has been agreed.
- Bounded: The purpose and theme of the new system are clear.
- Coherent: The requirements provide a consistent description of the essential characteristics of the new system.
- Acceptable: The requirements describe a system that is acceptable to the stakeholders.
- Addressed: Enough of the requirements have been addressed to satisfy the need for a new system in a way that is acceptable to the stakeholders.
- Fulfilled: The requirements that have been addressed fully satisfy the need for a new system.

**Associations**

scopes and constrains : Work

The Requirements scope and constrain the Work.

**Justification: Why Requirements?**

The requirements capture what the stakeholders want from the system. They define what the system must do, but not necessarily how it must do it. They describe the value the system will provide by addressing the opportunity and how the opportunity will be pursued by the production of a new software system. They also scope and constrain the work by defining what needs to be achieved.

The requirements are captured as a set of requirement items. The requirement items can be communicated and recorded in various forms and at various levels of detail. They may be communicated explicitly as a set of extensive requirements documents or more tacitly in the form of conversations and brainstorming sessions. The requirement items themselves are always documented and tracked. The documentation can take many forms and be as brief as a one-line user story or as comprehensive as a use case.

As the development of the system proceeds, the requirements evolve and are constantly re-prioritized and adjusted to reflect the changing needs of the stakeholders. Much that is implicit at first is made explicit later by adding more detailed requirement items such as well-defined quality characteristics and test cases. This allows the requirements to act as a verifiable specification for the software system. Regardless of how the requirement items are captured it is essential that the software system produced can be shown to successfully fulfill the requirements. This is why requirements play such an essential role in the testing of the system. As well as providing a definition of what needs to be achieved, they also allow tracking of what has been achieved. As the testing of each requirement item is completed it can be individually checked off as done, and the requirements as a whole can be looked at to see if the system produced sufficiently fulfills the requirements and whether or not work on the system is finished.

It is important that the overall state of the requirements is understood as well as the state of the individual requirement items. If the overall state of the requirements is not understood then it will be impossible to 1) tell when the system is finished, and 2) judge whether or not an individual requirement item is in the scope of the system.
Progressing the Requirements

During the development of a software system the requirements progress through several state changes. As shown in Figure 9, they are conceived, bounded, coherent, acceptable, addressed, and fulfilled. These states focus on the evolution of the team’s understanding of what the proposed system must do, from the conception of a new set of requirements as an initial idea for a new software system through their development to their fulfillment by the provision of a usable software system.

As shown in Figure 9, the requirements start in the conceived state when the need for a new software system has been agreed. The stakeholders can hold differing views on the overall meaning of the requirements. However, they all agree that there is a need for a new software system and a clear opportunity to be pursued.

Before too much time is spent collecting and detailing the individual requirement items the requirements as a whole must be bounded. To bound the requirements, the overall scope of the new system, the aspects of the opportunity to be addressed, and the mechanisms for managing and accepting new or changed requirement items all need to be established. In the bounded state there may still be inconsistencies or ambiguities between the individual requirement items. However, the stakeholders now have a shared understanding of the purpose of the new system and can tell whether or not a request qualifies as a requirement item. They also understand the mechanisms to be used to evolve the requirement items and remove the inconsistencies. Once the requirements are bounded there is a shared understanding of the scope of the new system and it is safe to start implementing the most important requirement items.

Further elicitation, refinement, analysis, negotiation, demonstration and review of the individual requirement items leads to a coherent set of requirements, one that clearly defines the essential characteristics of the new system. The requirement items continue to evolve as more is learnt about the new system and its impact on its stakeholders and environment. No matter how much the requirement items change, it is essential that they stay within the bounds of the original concept and that they remain coherent at all times.
The continued evolution of the requirements leads to the capture of an acceptable set of requirements, one that defines a system that will be acceptable to the stakeholders as, at least, an initial solution. The requirements may only describe a partial solution; however the solution described is of sufficient value that the stakeholders would accept it for operational use. The number of requirement items that need to be agreed for the requirements to be acceptable to the stakeholders can vary from one to many. When changing a mature system it may be acceptable to just address one important requirement item. When building a replacement system a large number of requirement items will need to be addressed.

As the individual requirement items are implemented and a usable system is evolved, there will come a time when enough requirements have been implemented for the new system to be worth releasing and using. In the addressed state the amount of requirements that have been addressed is sufficient for the resulting system to provide clear value to the stakeholders. If the resulting system provides a complete solution then the requirements may advance immediately to the fulfilled state.

Usually, when the addressed state is achieved the resulting system provides a valuable but incomplete solution. To fully address the opportunity, additional requirement items may have to be implemented. The shortfall may be because an incremental approach to the delivery of the system was selected, or because the missing requirements were difficult to identify before the system was made available for use.

In the fulfilled state enough of the requirement items have been implemented for the stakeholders to agree that the resulting system fully satisfies the need for a new system, and that there are no outstanding requirement items preventing the system from being considered complete.

Understanding the current and desired state of the requirements can help everyone understand what the system needs to do and how close to complete it is.

**Checking the Progress of the Requirements**

To help assess the state of the requirements and the progress being made towards their successful conclusion, the following checklists are provided:

**Table 10 – Checklist for Requirements**

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceived</td>
<td>The initial set of stakeholders agrees that a system is to be produced.</td>
</tr>
<tr>
<td></td>
<td>The stakeholders that will use the new system are identified.</td>
</tr>
<tr>
<td></td>
<td>The stakeholders that will fund the initial work on the new system are identified.</td>
</tr>
<tr>
<td></td>
<td>There is a clear opportunity for the new system to address.</td>
</tr>
<tr>
<td>Bounded</td>
<td>The stakeholders involved in developing the new system are identified.</td>
</tr>
<tr>
<td></td>
<td>The stakeholders agree on the purpose of the new system.</td>
</tr>
<tr>
<td></td>
<td>It is clear what success is for the new system.</td>
</tr>
<tr>
<td></td>
<td>The stakeholders have a shared understanding of the extent of the proposed solution.</td>
</tr>
<tr>
<td></td>
<td>The way the requirements will be described is agreed upon.</td>
</tr>
<tr>
<td></td>
<td>The mechanisms for managing the requirements are in place.</td>
</tr>
<tr>
<td></td>
<td>The prioritization scheme is clear.</td>
</tr>
<tr>
<td></td>
<td>Constraints are identified and considered.</td>
</tr>
<tr>
<td></td>
<td>Assumptions are clearly stated.</td>
</tr>
<tr>
<td>Coherent</td>
<td>The requirements are captured and shared with the team and the stakeholders.</td>
</tr>
<tr>
<td></td>
<td>The origin of the requirements is clear.</td>
</tr>
<tr>
<td></td>
<td>The rationale behind the requirements is clear.</td>
</tr>
<tr>
<td>State</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Acceptable</td>
<td>The stakeholders accept that the requirements describe an acceptable solution. The rate of change to the agreed requirements is relatively low and under control. The value provided by implementing the requirements is clear. The parts of the opportunity satisfied by the requirements are clear. The requirements are testable.</td>
</tr>
<tr>
<td>Addressed</td>
<td>Enough of the requirements are addressed for the resulting system to be acceptable to the stakeholders. The stakeholders accept the requirements as accurately reflecting what the system does and does not do. The set of requirement items implemented provide clear value to the stakeholders. The system implementing the requirements is accepted by the stakeholders as worth making operational.</td>
</tr>
<tr>
<td>Fulfilled</td>
<td>The stakeholders accept the requirements as accurately capturing what they require to fully satisfy the need for a new system. There are no outstanding requirement items preventing the system from being accepted as fully satisfying the requirements. The system is accepted by the stakeholders as fully satisfying the requirements.</td>
</tr>
</tbody>
</table>

8.3.2.2 Software System

Description

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

A software system can be part of a larger software, hardware, business or social solution.

States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Selected</td>
<td>An architecture has been selected that addresses the key technical risks and any applicable organizational constraints.</td>
</tr>
<tr>
<td>Demonstrable</td>
<td>An executable version of the system is available that demonstrates the architecture is fit for purpose and supports testing.</td>
</tr>
<tr>
<td>Usable</td>
<td>The system is usable and demonstrates all of the quality characteristics of an operational system.</td>
</tr>
<tr>
<td>Ready</td>
<td>The system (as a whole) has been accepted for deployment in a live environment.</td>
</tr>
<tr>
<td>Operational</td>
<td>The system is in use in a live environment.</td>
</tr>
<tr>
<td>Retired</td>
<td>The system is no longer supported.</td>
</tr>
</tbody>
</table>
Associations

helps to address : Opportunity  
fulfills : Requirements

Software System helps to address Opportunity.
Software Systems fulfills Requirements.

Justification: Why Software System?

Essence uses the term software system rather than software because software engineering results in more than just a piece of software. Whilst the value may well come from the software, a working software system depends on the combination of software, hardware and data to fulfill the requirements.

Progressing the Software System

The life-cycle of a software system is hard to define as there can be many releases of a software system. These releases can be worked on and used in parallel. For example one team can be working on the development of release 3, whilst another team is making small changes to release 2, and a third team is providing support for those people still using release 1. If we treat this software system as one entity what state is it in?

To keep things simple, Essence treats each major release as a separate software system; one that is built, released, updated, and eventually retired. A major release encompasses significant changes to the purpose, usage, or architecture of a software system. It can encompass many minor releases including internal releases produced for testing purposes, and external releases produced to support incremental delivery or bug fixes. In the example above the second team would be producing a series of minor releases (2.1, 2.2, 2.3, etc.) of their software system to allow the delivery of their small changes.

During its development a software system progresses through several state changes. As shown in Figure 8, they are architecture selected, demonstrable, usable, ready, operational and retired. These states provide points of stability on a software system’s journey from its conception to its eventual retirement indicating (1) when the architecture is selected,
(2) when a demonstrable system is produced to prove the architecture and enable testing to start, (3) when the system is extended and improved so that it becomes usable, (4) when the usable system is enhanced until it is accepted as ready for deployment, (5) when the system is made available to the stakeholders who use it and made operational, and finally, (6) when the system itself is retired and its support is withdrawn. These states can be applied to the initial release of the software system or any subsequent modification or replacement.

As indicated in Figure 10, the first thing to do for any major software system release is to make sure that there is an appropriate architecture available; one that complies with any applicable organizational constraints and addresses the key technical risks facing the new system. Achieving this may require the creation of a brand new architecture, the modification of an existing architecture, the selection of an existing architecture, or the simple re-use of whatever is already in place. Regardless of the approach taken, the result is that the system progresses to the architecture selected state.

Once the architecture had been selected, it must be shown to be fit-for-purpose by building and testing a demonstrable version of the system. It is not sufficient to just present a set of rolling screen-shots or a stand-alone version of a multi-user system. The system needs to be truly demonstrable exercising all of the significant characteristics of the selected architecture. It must also be capable of supporting both functional and non-functional testing.

The demonstrable system is then evolved to become usable by adding more functionality, and fixing defects. Once the system has achieved the usable state, it has all the qualities desired of an operational system. If it implements a sufficient amount of the requirements, if it provides sufficient business value, and if there is an appropriate window of opportunity for its deployment, then it can be considered to be ready for operational use.

Although, a useable system has the potential to be an operational system, there are still a few essential steps to be performed before it is ready. The system has to be accepted for use by the stakeholders, and it has to be prepared for deployment in the live environment. In this state, the system is typically supplemented with installation guidance, training materials and actual training for system operation.

The system is made operational when it is installed for real use within the live environment. It is now being used to generate value and provide benefit to its stakeholders.

Even after the software system has been made operational, development work can still continue. This may be as part of the plans for the incremental delivery of the system or, as is more common, a response to defects and problems occurring during the deployment and operation of the system. Support and maintenance continue until the software system is retired and its support is withdrawn. This may be because 1) the software system has been completely replaced by a later generation, 2) the software system no longer has any users or, 3) it does not make business sense to continue to support it.

During the development of a major release many minor releases are often produced. For example, many teams using an iterative approach produce a new release during every iteration whilst they keep their software system continuously in a usable, and therefore potentially shippable, state. It is then the stakeholder representatives who decide whether it is ready to be made operational. Obviously, this approach is not always possible, particularly if major architectural changes are required as these often render the system unusable for a significant period of time.

Understanding the current and desired states of a software system helps everyone understand when a system is ready, what kinds of changes can be realistically made to the system, and what kinds of work should be left to a later generation of the software system.

**Checking the Progress of the Software System**

To help assess the state of a software system and the progress being made towards its successful operation, the following checklist items are provided:

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Selected</td>
<td>The criteria to be used when selecting the architecture have been agreed on.</td>
</tr>
<tr>
<td></td>
<td>Hardware platforms have been identified.</td>
</tr>
<tr>
<td></td>
<td>Programming languages and technologies to be used have been selected.</td>
</tr>
</tbody>
</table>
| System boundary is known.  
Significant decisions about the organization of the system have been made.  
Buy, build and reuse decisions have been made. |
|---|
| Demonstrable | Key architectural characteristics have been demonstrated.  
The system can be exercised and its performance can be measured.  
Critical hardware configurations have been demonstrated.  
Critical interfaces have been demonstrated.  
The integration with other existing systems has been demonstrated.  
The relevant stakeholders agree that the demonstrated architecture is appropriate. |
| Usable | The system can be operated by stakeholders who use it.  
The functionality provided by the system has been tested.  
The performance of the system is acceptable to the stakeholders.  
Defect levels are acceptable to the stakeholders.  
The system is fully documented.  
Release content is known.  
The added value provided by the system is clear. |
| Ready | Installation and other user documentation are available.  
The stakeholder representatives accept the system as fit-for-purpose.  
The stakeholder representatives want to make the system operational.  
Operational support is in place. |
| Operational | The system has been made available to the stakeholders intended to use it.  
At least one example of the system is fully operational.  
The system is fully supported to the agreed service levels. |
| Retired | The system has been replaced or discontinued.  
The system is no longer supported.  
There are no “official” stakeholders who still use the system.  
Updates to the system will no longer be produced. |

### 8.3.3 Activity Spaces

The solution area of concern contains six activity spaces that cover the capturing of the requirements and the development of the software system.

#### 8.3.3.1 Understand the Requirements

**Description**

Establish a shared understanding of what the system to be produced must do.

Understand the requirements to:
- Scope the system.
- Understand how the system will generate value.
- Agree on what the system will do.
- Identify specific ways of using and testing the system.
- Drive the development of the system.

**Input:** Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working  
**Completion Criteria:** Requirements::Conceived, Requirements::Bounded, Requirements::Coherent

### 8.3.3.2 Shape the System

**Description**  
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 overall design and architecting of the system to be produced.

Shape the system to:
- Structure the system and identify the key system elements.
- Assign requirements to elements of the system.
- Ensure that the architecture is suitably robust and flexible.

**Input:** Stakeholders, Opportunity, Requirements, Software System, Work, Way-of-Working  
**Completion Criteria:** Requirements::Sufficient, Software System::Architecture Selected

### 8.3.3.3 Implement the System

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

Implement the system to:
- Create a working system.
- Develop, integrate and test the system elements.
- Increase the number of requirements implemented.
- Fix defects.
- Improve the system

**Input:** Requirements, Software System, Way-of-Working  
**Completion Criteria:** Software System::Demonstrable, Software System::Usable, Software System::Ready

### 8.3.3.4 Test the System

**Description**  
Verify that the system produced meets the stakeholders’ requirements.

Test the system to:
- Verify that the software system matches the requirements
- Identify any defects in the software system.

**Input:** Requirements, Software System, Way-of-Working
Completion Criteria: Requirements::Sufficient, Requirements::Fulfilled, Software System::Demonstrable, Software System::Usable, Software System::Ready

### 8.3.3.5 Deploy the System

**Description**
Take the tested system and make it available for use outside the development team.

Deploy the system to:
- Package the software system up for delivery to the live environment.
- Make the software system operational.

**Input:** Stakeholders, Software System, Way-of-Working

**Completion Criteria:** Software System::Operational

### 8.3.3.6 Operate the System

**Description**
Support the use of the software system in the live environment.

Operate the system to:
- Maintain service levels.
- Support the stakeholders who use the system.
- Support the stakeholders who deploy, operate, and help support the system.

**Input:** Stakeholders, Opportunity, Requirements, Software System, Way-of-Working

**Completion Criteria:** Software System::Retired

### 8.3.4 Competencies

#### 8.3.4.1 Analysis

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

The analysis competency is the deductive ability to understand the situation, context, concepts and problems, identify appropriate high-level solutions, and evaluate and draw conclusions by applying logical thinking.

People with the analytical competency help the team to:
- Identify and understand needs and opportunities.
- Get to know the root causes of the problems.
- Capture, understand and communicate requirements.
- Create and agree on specifications and models.
- Visualize solutions and understand their impact.

**Essential skills include:**
- Verbal and written communication
- Ability to observe, understand, and record details
• Agreement facilitation
• Requirements capture
• Ability to separate the whole into its component parts
• Ability to see the whole by looking at what is required

This competency can be provided by the customer representatives, product owners, business analysts, requirement specialists or developers on the team.

Competency Levels

Level 1 – Assists
Demonstrates a basic understanding of the concepts and can follow instructions.

Level 2 – Applies
Able to apply the concepts in simple contexts by routinely applying the experience gained so far.

Level 3 – Masters
Able to apply the concepts in most contexts and has the experience to work without supervision.

Level 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.

Level 5 - Innovates
A recognized expert, able to extend the concepts to new contexts and inspire others.

Justification: Why Analysis?

Analysis is an examination of a system including its environment, its elements, and their relations. It is performed in order to gather, manage and analyze large and complex amounts of information and data and make sense of it. It is more than just the separation of a whole into its component parts as it involves the resolution of complex expressions into simpler or more basic ones, and the clarification of the purpose of a system by an explanation of its use.

When developing software it is essential that the current situation is analyzed and the correct requirements identified for the new system. The requirements themselves must also be analyzed to make sure that they are, amongst other things, practical, achievable and appropriately sized to drive the system’s development. The analysis competency encapsulates the abilities needed to successfully define the system to be built.

8.3.4.2 Development

Description

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

The development competency is the mental ability to conceive and produce a software system, or one of its elements, for a specific function or end. It enables a team to produce software systems that meet the requirements.

People with the development competency help the team to:

• Design and code software systems
• Formulate and/or evaluate strategies for choosing an appropriate design pattern or for combining various design patters
• Design and leverage technical solutions
• Troubleshoot and resolve coding problems

Essential skills include:

• Knowledge of technology
• Programming
• Knowledge of programming languages
• Critical thinking
• Re-factoring
• Design

This competency can be provided by the programmers, coders, designers or architects on the team.

**Competency Levels**

Level 1 – Assists
Demonstrates a basic understanding of the concepts and can follow instructions.

Level 2 – Applies
Able to apply the concepts in simple contexts by routinely applying the experience gained so far.

Level 3 – Masters
Able to apply the concepts in most contexts and has the experience to work without supervision.

Level 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.

Level 5 - Innovates
A recognized expert, able to extend the concepts to new contexts and inspire others.

**Justification: Why Development?**

Developing a software system is a complex mental activity requiring the ability to exploit all the knowledge about the opportunity, stakeholder’s needs, company’s business, the technology used and balance them by creating an appropriate solution. It requires a combination of talent, experience, knowledge and programming skills in order to develop the right solution.

The development competency is about solving complex problems and producing effective software systems. It lies in the observing, the sense-making of and representing the system as others expect it to see it, that is, as effective and functional and easy to use. All this in turn requires the ability to imagine and visualize code and structure it in a way so that it is easy to understand and maintain.

**8.3.4.3 Testing**

**Description**

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

The testing competency is an observational, comparative, detective and destructive ability that enables the system to be tested.

People with the testing competency help the team to:

• Test the system
• Create the correct tests to efficiently verify the requirements
• Decide what, when and how to test
• Evaluate whether the system meets the requirements
• Find defects and understand the quality of the system produced.

Essential skills include:

• Keen observation
• Exploratory and destructive thinking
• Inquisitive mind
• Attention to detail

This competency can be provided by specialist individuals or other team members such as customers, users, analysts, developers or other stakeholders.
Competency Levels

Level 1 – Assists: Demonstrates a basic understanding of the concepts and can follow instructions.
Level 2 – Applies: Able to apply the concepts in simple contexts by routinely applying the experience gained so far.
Level 3 – Masters: Able to apply the concepts in most contexts and has the experience to work without supervision.
Level 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.
Level 5 – Innovates: A recognized expert, able to extend the concepts to new contexts and inspire others.

Justification: Why Testing?

When developing software it is essential to test that the system meets the requirements and demonstrate that it is fit for purpose. The ability to conceive and undertake testing is essential throughout the evolution of a system, and is an essential complement to the team’s analysis, design and programming capabilities.

The testing competency encapsulates the ability to conceive and execute tests to demonstrate that the system is fit for purpose, usable, meets one or more of its requirements and constitutes an appropriate solution to the stakeholders needs.

8.4 The Endeavor Area of Concern

8.4.1 Introduction

This area of concern contains everything to do with the team, and the way that they approach their work.

Software engineering is a significant endeavor that typically takes many weeks to complete, affects many different people (the stakeholders) and involves a development team (rather than a single developer). Any practical method must describe a set of practices to effectively plan, lead and monitor the efforts of the team.

8.4.2 Alphas

The endeavor area of concern contains the following Alphas:

- Team
- Work
- Way-of-Working

8.4.2.1 Team

Description

Team: The group of people actively engaged in the development, maintenance, delivery and support of a specific software system.

The team plans and performs the work needed to create, update and/or change the software system.

States

Seeded: The team’s mission is clear and the know-how needed to grow the team is in place.
Formed: The team has been populated with enough committed people to start the mission.
Collaborating: The team members are working together as one unit.
Performing: The team is working effectively and efficiently.
Adjourned: The team is no longer accountable for carrying out its mission.
Associations
produces : Software System  Team produces Software System.
performs and plans : Work  Team performs and plans Work.

Justification: Why Team?
Software engineering is a team sport involving the collaborative application of many different competencies and skills. The effectiveness of a team has a profound effect on the success of any software engineering endeavor. To achieve high performance, team members should reflect on how well they work together, and relate this to their potential and effectiveness in achieving their mission.

Normally a team consists of several people. Occasionally, however, work may be undertaken by a single individual creating software purely for their own use and entertainment. A team requires at least two people, but the guidance provided by the Team Alpha can also be used to help single individuals when creating software.

Progressing the Team
Teams evolve during their time together and progress through several state changes. As shown in Figure 9, the states are seeded, formed, collaborating, performing, and adjourned. They communicate the progression of a software team on the journey from initial conception to the completion of the mission indicating (1) when the team is seeded and the individuals start to join the team (2) when the team is formed to start the mission, (3) when the individuals start collaborating effectively and truly become a team, (4) when the team is performing and achieves a crucial level of efficiency and productivity, and (5) when the team is adjourned after completing its mission.

As shown in Figure 11, the team is first seeded. This implies defining the mission, deciding on recruitment for the necessary skills, capabilities and responsibilities, and making sure that the conditions are right for an effective group to come together. As the team is formed, the people in the group, and those joining it, bring the necessary skills and experience to the team. The group becomes a team as the people begin to see how they can contribute to the work at hand. As they discover and take account of each others’ capabilities, they start collaborating effectively and make progress towards completing their mission.

At its peak of performing, the team shares a way of working, and plays to its strengths to complete its mission effectively and efficiently. The performing team easily adapts to the changing context and takes appropriate measures. If a number of people join or leave the team, or the context of the mission changes, it may revert to a previous state. Finally, if the team has no further goals or missions to complete, it is adjourned.

It is important to understand the current state of the team so that suitable practices can be used to address the issues and impediments being faced, and to ensure that the team focuses on working effectively and efficiently.
The team's mission is clear and the know-how needed to grow the team is in place.

The team has been populated with enough committed people to start the mission.

The team members are working together as one unit.

The team is working effectively and efficiently.

The team is no longer accountable for carrying out its mission.

*Figure 11 – The states of the Team*

**Checking the Progress of the Team**

To help assess the state of a team and its progress, the following checklists are provided:

*Table 12 – Checklist for Team*

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeded</td>
<td>The team mission has been defined in terms of the opportunities and outcomes.</td>
</tr>
<tr>
<td></td>
<td>Constraints on the team's operation are known.</td>
</tr>
<tr>
<td></td>
<td>Mechanisms to grow the team are in place.</td>
</tr>
<tr>
<td></td>
<td>The composition of the team is defined.</td>
</tr>
<tr>
<td></td>
<td>Any constraints on where and how the work is carried out are defined.</td>
</tr>
<tr>
<td></td>
<td>The team's responsibilities are outlined.</td>
</tr>
<tr>
<td></td>
<td>The level of team commitment is clear.</td>
</tr>
<tr>
<td></td>
<td>Required competencies are identified.</td>
</tr>
<tr>
<td></td>
<td>The team size is determined.</td>
</tr>
<tr>
<td></td>
<td>Governance rules are defined.</td>
</tr>
<tr>
<td></td>
<td>Leadership model is selected.</td>
</tr>
<tr>
<td>Formed</td>
<td>Individual responsibilities are understood.</td>
</tr>
</tbody>
</table>
Enough team members have been recruited to enable the work to progress.
Every team member understands how the team is organized and what their individual role is.
All team members understand how to perform their work.
The team members have met (perhaps virtually) and are beginning to get to know each other
The team members understand their responsibilities and how they align with their competencies.
Team members are accepting work.
Any external collaborators (organizations, teams and individuals) are identified.
Team communication mechanisms have been defined.
Each team member commits to working on the team as defined.

Collaborating
The team is working as one cohesive unit.
Communication within the team is open and honest.
The team is focused on achieving the team mission.
The team members know each other.

Performing
The team consistently meets its commitments.
The team continuously adapts to the changing context.
The team identifies and addresses problems without outside help.
Effective progress is being achieved with minimal avoidable backtracking and reworking.
Wasted work, and the potential for wasted work are continuously eliminated.

Adjourned
The team responsibilities have been handed over or fulfilled.
The team members are available for assignment to other teams.
No further effort is being put in by the team to complete the mission.

8.4.2.2 Work

Description
Work: Activity involving mental or physical effort done in order to achieve a result.

In the context of software engineering, work is everything that the team does to meet the goals of producing a software system matching the requirement and addressing the opportunity presented by the stakeholders. The work is guided by the practices that make up the team’s way-of-working.

States
Initiated
The work has been requested.

Prepared
All pre-conditions for starting the work have been met.

Started
The work is proceeding.

Under Control
The work is going well, risks are under control, and productivity levels are sufficient to achieve a satisfactory result.

Concluded
The work to produce the results has been concluded.

Closed
All remaining housekeeping tasks have been completed and the work has been officially closed.
Associations

updates and changes: Software System  Work updates and changes Software System.
set up to address : Opportunity  Work set up to address Opportunity.

Justification: Why Work?
The ability of team members to co-ordinate, organize, estimate, complete, and share their work has a profound effect on meeting their commitments and delivering value to their stakeholders. Team members need to understand how to carry out their work, and how to recognize when the work is going well.

Progressing the Work
During the development of a software system the work progresses through several state changes. As shown in Figure 10, they are initiated, prepared, started, under control, concluded, and closed. These states provide points of stability in the progression of the work indicating when the work is initiated and prepared, when the team is assembled and the work is started and brought under control, when the results are achieved and the development work is concluded, and finally, when the work itself is closed and all loose ends and outstanding work items are addressed.

As indicated in Figure 12, the work is first initiated. This implies that someone defines the desired result, and makes sure that the conditions are right for the work to be performed. If the work is not successfully initiated, it will never be

![Diagram of Work States](image)

**Figure 12 – The states of the Work**

progressed and assigned to a team. As the work is prepared, commitments are made, funding and resources are secured, the work is organized, appropriate governance policies and procedures are put in place, and priorities, constraints and impediments are understood. Once all the pre-conditions for starting the work are addressed, the team gets the go-ahead to get the real work started. The team starts to complete the individual work items, and builds evidence showing that the work is under control.

There are many practices that can be used to help organize and co-ordinate the work including SCRUM, Kanban,
PMBoK, PRINCE2, Task Boards and many, many more. These typically involve breaking the work down into:

1. Smaller, more bite sized work items that can be completed one-by-one such as work packages, and tasks.
2. One or more clearly defined work periods such as phases, stages, iterations, or sprints.

The level, depth and extent of the work breakdown depends on the style and complexity of the work and on the specific practices the team selects to help them co-ordinate, monitor, control and undertake the work.

If the team has their work under control then there will be concrete evidence that:

1. The work is going well.
2. The risks threatening a successful conclusion to the work are under control as the impact if they occur and/or as the likelihood of them occurring have been reduced to acceptable levels.
3. The team’s productivity levels are sufficient to achieve satisfactory results within the time, budget and any other constraints that have been placed upon the work.

Typically, once the work has been concluded and the results have been accepted by the relevant stakeholders, there remain some final housekeeping and wrap up activities to be completed before the work itself can be closed.

If, for any reason, the work is not going well, then it may be halted, abandoned or reverted to a previous state. If the work is abandoned once it is started, it should still be properly closed even though it has not managed to pass through the concluded state.

Understanding the current and desired state of the work can help the team to balance their activities, make the correct investment decisions, nurture the work that is going well, and help or cancel the work that is going badly.

**Checking the Progress of the Work**

To help assess the state of the work and the progress being made towards its successful conclusion, the following checklists are provided:

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
</table>
| Initiated | The result required of the work being initiated is clear.  
Any constraints on the work’s performance are clearly identified.  
The stakeholders that will fund the work are known.  
The initiator of the work is clearly identified.  
The stakeholders that will accept the results are known.  
The source of funding is clear.  
The priority of the work is clear. |
| Prepared | Commitment is made.  
Cost and effort of the work are estimated.  
Resource availability is understood.  
Governance policies and procedures are clear.  
Risk exposure is understood.  
Acceptance criteria are defined and agreed with client.  
The work is broken down sufficiently for productive work to start.  
Tasks have been identified and prioritized by the team and stakeholders. |
<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
</table>
| Started | Development work has been started.  
Work progress is monitored.  
The work is being broken down into actionable work items with clear definitions of done.  
Team members are accepting and progressing tasks. |
| Under Control | Tasks are being completed.  
Unplanned work is under control.  
Risks are under control as the impact if they occur and the likelihood of them occurring have been reduced to acceptable levels.  
Estimates are revised to reflect the team’s performance.  
Measures are available to show progress and velocity.  
Re-work is under control.  
Tasks are consistently completed on time and within their estimates. |
| Concluded | All outstanding tasks are administrative housekeeping or related to preparing the next piece of work.  
Work results have been achieved.  
The stakeholder(s) has accepted the resulting software system. |
| Closed | Lessons learned have been itemized, recorded and discussed.  
Metrics have been made available.  
Everything has been archived.  
The budget has been reconciled and closed.  
The team has been released.  
There are no outstanding, uncompleted tasks. |

**8.4.2.3 Way-of-Working**

**Description**

*Way-of-Working*: The tailored set of practices and tools used by a team to guide and support their work.

The team evolves their way of working alongside their understanding of their mission and their working environment. As their work proceeds they continually reflect on their way of working and adapt it to their current context, if necessary.

**States**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principles Established</td>
<td>The principles, and constraints, that shape the way-of-working are established.</td>
</tr>
<tr>
<td>Foundation Established</td>
<td>The key practices, and tools, that form the foundation of the way of working are selected and ready for use.</td>
</tr>
<tr>
<td>In Use</td>
<td>Some members of the team are using, and adapting, the way-of-working.</td>
</tr>
</tbody>
</table>
In Place: All team members are using the way of working to accomplish their work.
Working well: The team’s way of working is working well for the team.
Retired: The way of working is no longer in use by the team.

**Associations**

**Justification: Why Way-of-Working?**
Software engineering is a team sport, one that requires the whole team to collaborate effectively regardless of how the team is organized. They need to agree on a way of working that will support collaboration guide them throughout the software engineering endeavor.

The way of working:
- Is key to enabling a team to work together effectively.
- Focuses the team on how they will collaborate to ensure success.
- Enables the work to be planned and controlled.
- Helps the team, and their associated stakeholders, to successfully fulfill their responsibilities.

---

The states of the Way-of-Working:

- **Principles Established**: The principles, and constraints, that shape the way-of-working are established.
- **Foundation Established**: The key practices, and tools, that form the foundation of the way of working are selected and ready for use.
- **In Use**: Some members of the team are using, and adapting, the way-of-working.
- **In Place**: All team members are using the way-of-working to accomplish their tasks.
- **Working well**: The way-of-working is working well for the team.
- **Retired**: The way-of-working is no longer in use by the team.

**Figure 13 – The states of the Way-of-Working**

**Progressing the Way-of-Working**
During the course of a software engineering endeavor the way of working progresses through several state changes. As presented in Figure 13, they are *principles established, foundation established, in use, in place, working well, and retired*. These states focus on the way a team establishes an effective way-of-working indicating (1) when the principles and constraints that shape the way-of-working are established, (2) when a minimal number of key practices and tools have
been identified and integrated to establish a foundation for the evolution of the team’s way-of-working, (3) when the chosen way of working is in use by the team, (4) when a team’s way of working is in place and in use by the whole team (5) when it is working well, and (6) when the way of working has been retired and is no longer in use by the team. Examples of principles and constraints could be how far into the future you plan, governance policies, how decisions are made, and how the work in broken down.

There are many ways of working that the team could adopt to meet their objectives and establish their approach to software engineering. As shown in Figure 13, the first step in adopting a new way-of-working, or adapting an existing way-of-working, is to understand the team’s working environment and establish the principles that will guide their selection of appropriate practices and tools. This includes identifying the constraints governing the selection of the team’s practices and tools and understanding the practices and tools that the team, and their stakeholders, are already using or are required to use.

It is not enough to just understand the principles and constraints that will inform the team's way of working. These must be agreed with, and actively supported by, the team and its stakeholders. Once the principles are established the team is ready to start selecting the practices and tools that will form their way-of-working.

To establish a natural way of working the focus should first be on the key practices and tools; those that bring the team together, enable communication among the team members, support collaborative working and are essential to the success of the team. However, these practices and tools act as the foundation for the team’s way-of-working. Before the foundation can be assembled it is important to understand the gaps between the practices and tools needed by the team and the practices, and tools immediately available to the team. This enables the activities needed to fill these gaps to be planned.

Once the key practices and tools are integrated then the way-of-working’s foundation is established and the way-of-working is ready to be trialed by the team. It will however be continuously adapted as the work progresses, and additional practices and tools will be added as the team inspects their way-of-working and adapts it to meet their changing circumstances.

Rather than spending more time tailoring or tuning the way-of-working it is important that the team puts it into use as soon as possible. The way-of-working is in use as soon as any of the team members are using and adapting it as part of completing their work. As more and more of the team start to use and benefit from the way-of-working its usage will grow until it is firmly in place and all the team members are using it to accomplish their work. Some team members may still need help to understand certain aspects of the team's way of working and to make effective progress, but the way of working is now the normal way for the team to develop software.

As the team progresses through the work, the way of working will become embedded in their activities and collaborations to such an extent that its use, inspection and adaptation are all seen as a natural part of the way the team works. The way-of-working is working well once it has stabilized and all team members are making progress as planned by using and adapting it to suit their current working environment. Finally, when the way of working is no longer in use by the team, it is retired.

Understanding the current and desired state of the team's way of working helps a team to continually improve their performance, and adapt quickly and effectively to change.

Checking the Progress of the Way-of-Working

To help assess the current status of the way of working, the following checklists are provided:

<table>
<thead>
<tr>
<th>State</th>
<th>Checklist</th>
</tr>
</thead>
</table>
| Principles Established | Principles and constraints are committed to by the team.  
Principles and constraints are agreed to by the stakeholders.  
The tool needs of the work and its stakeholders are agreed.  
A recommendation for the approach to be taken is available.  
The context within which the team will operate is understood. |

Table 14 – Checklist for Way-of-Working
The constraints that apply to the selection, acquisition and use of practices and tools are known.

**Foundation Established**
- The key practices and tools that form the foundation of the way-of-working are selected.
- Enough practices for work to start are agreed to by the team.
- All non-negotiable practices and tools have been identified.
- The gaps that exist between the practices and tools that are needed and the practices and tools that are available have been analyzed and understood.
- The capability gaps that exist between what is needed to execute the desired way of working and the capability levels of the team have been analyzed and understood.
- The selected practices and tools have been integrated to form a usable way-of-working.

**In Use**
- The practices and tools are being used to do real work.
- The use of the practices and tools selected are regularly inspected.
- The practices and tools are being adapted to the team’s context.
- The use of the practices and tools is supported by the team.
- Procedures are in place to handle feedback on the team’s way of working.
- The practices and tools support team communication and collaboration.

**In Place**
- The practices and tools are being used by the whole team to perform their work.
- All team members have access to the practices and tools required to do their work.
- The whole team is involved in the inspection and adaptation of the way-of-working.

**Working well**
- Team members are making progress as planned by using and adapting the way-of-working to suit their current context.
- The team naturally applies the practices without thinking about them.
- The tools naturally support the way that the team works.
- The team continually tunes their use of the practices and tools.

**Retired**
- The team's way of working is no longer being used.
- Lessons learned are shared for future use.

### 8.4.3 Activity Spaces

The endeavor area of concern contains five activity spaces that cover the formation and support of the team, and planning and co-coordinating the work in-line with the way of working.

#### 8.4.3.1 Prepare to do the Work

**Description**

Set up the team and its working environment. Understand and commit to completing the work.

Prepare to do the work to:

- Put the initial plans in place.
- Establish the initial way of working.
- Assemble and motivate the initial project team.
- Secure funding and resources.

**Input:** Stakeholders, Opportunity, Requirements  
**Completion Criteria:** Team::Seeded, Way of Working::Principles Established, Way of Working::Foundation Established, Work::Initiated, Work::Prepared

### 8.4.3.2 Coordinate Activity

**Description**
Co-ordinate and direct the team’s work. This includes all ongoing planning and re-planning of the work, and adding any additional resources needed to complete the formation of the team.

Coordinate activity to:
- Select and prioritize work.
- Adapt plans to reflect results.
- Get the right people on the team.
- Ensure that objectives are met.
- Handle change.

**Input:** Requirements, Team, Work, Way of Working  
**Completion Criteria:** Team::Formed, Work::Started, Work::Under Control

### 8.4.3.3 Support the Team

**Description**
Help the team members to help themselves, collaborate and improve their way of working.

Support the team to:
- Improve team working.
- Overcome any obstacles.
- Improve ways of working.

**Input:** Team, Work, Way of Working  
**Completion Criteria:** Team::Collaborating, Way of Working::In Use, Way of Working::In Place

### 8.4.3.4 Track Progress

**Description**
Measure and assess the progress made by the team.

Track progress to:
- Evaluate the results of work done.
- Measure progress.
- Identify impediments.

**Input:** Requirements, Team, Work, Way of Working  
**Completion Criteria:** Team::Performing, Way of Working::Working Well, Work::Under Control, Work::Concluded
8.4.3.5 Stop the Work

Description
Shut-down the software engineering endeavor and handover the team’s responsibilities.

Stop the work to:

- Close the work.
- Handover any outstanding responsibilities.
- Handover any outstanding work items.
- Stand down the team.
- Archive all work done.

Input: Requirements, Team, Work, Way of Working
Completion Criteria: Team::Adjourned, Way of Working::Retired, Work::Closed

8.4.4 Competencies

8.4.4.1 Leadership

Description
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.

People with the leadership competency help the team to:

- Inspire people to do their work
- Make sure that all team members are effective in their assignments
- Make and meet their commitments
- Resolve any impediments or issues holding up the team's work
- Interact with stakeholders to shape priorities, report progress and respond to challenges.

Essential skills include:

- Inspiration
- Motivation
- Negotiation
- Communication
- Decision making

This competency is sometimes provided by a Scrum Master, an appointed team leader, the more experienced members of the team, or a dedicated project manager.

Competency Levels

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

**Justification: Why Leadership?**

Software engineering is a complex endeavor typically involving teams of people dedicated to delivering an appropriate solution to extended networks of customers, users and other stakeholders. It is essential that everybody is focused, inspired and motivated towards achieving the same goals.

Within the software engineering kernel, the leadership competency is the ability to radiate enthusiasm, energy, trustworthiness, confidentiality and direction. The people with this competency guide and help the team to a successful conclusion, one that satisfies the needs of the stakeholders, within acceptable time and cost constraints.

### 8.4.4.2 Management

**Description**

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

The management competency is the administrative and organizational ability that enables the right things to be done at the right time to maximize a team’s chances of success.

Management helps the team to:

- Proactively manage risks
- Account for time and money spent
- Interact with stakeholders to report progress
- Coordinate and plan activities

**Essential skills include:**

- Communication
- Administration
- Organization
- Resource planning
- Financial reporting

This competency can be provided by the team members themselves, a team leader, a lead developer, a project management office or a professional project manager.

**Competency Levels**

| Level 1 – Assists | Demonstrates a basic understanding of the concepts and can follow instructions. |
| Level 2 – Applies | Able to apply the concepts in simple contexts by routinely applying the experience gained so far. |
| Level 3 – Masters | Able to apply the concepts in most contexts and has the experience to work without supervision. |
| Level 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. |
| Level 5 - Innovates | A recognized expert, able to extend the concepts to new contexts and inspire others. |

**Justification: Why Management?**

Software engineering is a complex endeavor that requires the organization and coordination of many people and other resources. It needs the team to possess the ability to track progress, organize facilities and events, co-ordinate all the work, and integrate into the structure of the owning organization. The management competency encapsulates the abilities needed to be able to coordinate and track the work done by the team.