Guide To
PROJECT MANAGEMENT
Life Cycles

SUCCESS, STARTING WITH THE END IN MIND

William H. Volpé III, PMP

Guide to Project Management Life Cycles

William H. Volpe III, PMP

Author's Note

This book consists of details related to Project Management Life Cycles, their selection and implementation according to the nature of the project. A detailed description of each project management life cycle along with their use is incorporated for the better understanding of the reader.

Published by The Volpé Consortium, Inc., Texas

Printed in the United States of America.

For more information, please write:

The Volpé Consortium, Inc.

Email: contact@thevolpeconsortium.com

or call 1(512) 632 5172

Visit us online at: www.thevolpeconsortium.com

Abstract

The purpose of the initiation and completion of any project is to introduce new services and products in this technological business world. However, these projects must produce results within the given time frame and budget of the effort. As a project manager, one must be familiar with different project management life cycles in order to choose the best one for successful completion of their project.

In this book it is explained how different project life cycles have helped to achieve high quality results effectively and efficiently within the triple constraints of project management. Contained are detailed descriptions of different project management life cycles that include their history and some of their best uses in project management. This book also details the unique features of all frameworks and enables the reader to grasp the differences among them.

Table of Contents

Introduction

With the rapid advancing in competition and globalization, industries are moving towards project based development rather than the typical processes used in every organization. Repetitive operations, also known as processes, were once useful for all kinds of organizations. However, since the advent of technology and rapid advancement in the business environment, the consumer's demand and expectations are changing at a fast pace.

Companies seek to meet these expectations at all cost and they opt for products and services that are unique in nature and will eventually satisfy the consumer. Coming up with unique ideas requires exceptional skills and abilities, which is not as easy as it sounds. The customer's taste, economic factors, and political issues to name a few have great impact on how the product will eventually turn out. Moreover, producing a unique product cannot be an on-going or never ending process. It is a unique one time effort known as a "project."

What is a Project?

Projects, basically, are unique and temporary endeavors undertaken to achieve a specific goal, to produce a distinctive product, or provide a service that has never been offered before. Projects have defined beginning and end times and are under time, budget, and deliverable constraints (Institute, 2008). The temporary nature of the project does not always mean that the projects are of shorter durations. It implies that they are bound to end sooner or later. They can be as short as a week or as long as 10 years or longer.

Project and processes are different because a process is repetitive by nature and is an ongoing activity, whereas projects are one time unique endeavors (Stanleigh, n.d.). However, the project is completed following a process that has been designed for project execution. It consists of predefined steps that are carried out to meet the client's requirements fully (Current ITS Projects, n.d.). Projects can be executed at all levels of organizations. They are the means for implementation of organizational strategy and therefore they are planned and executed with great care.

Organizations place great importance on the successful execution of projects, because the higher the success rate of projects, the better the company image is. The time limit assigned to each project can vary according to the nature and requirements of the project. At times, companies join hands with other organizations to complete the project on time or to obtain additional resources (William R. Duncan, 1996, p. 4). For example, a software

development company may develop a particular software on their own and for software quality assurance they may hire another company to oversee the process. This is also known as “outsourcing.”

Projects also promote the concept of cross functional decision making. Different people with different skills and abilities are brought together to work towards a mutual goal. Therefore, different projects with different life cycles are all required to be completed effectively and efficiently.

The five main characteristics of projects are listed below (University).

- Projects are temporary endeavors that have a defined start and end date.
- Projects help in the design and execution of organizational strategy.
- Projects are the cause of new products and services introduced with the help of specific resources.
- Projects promote cross functionality in an organization. A team is formed to work on a project that is disbanded after completion.

Project Management

Whether it is the introduction of a unique product or a regeneration of an existing product, organizations are opting for project management to improve their worth in competing markets. Hence, project management can be described as the set of skills, knowledge, tools, and techniques needed to achieve the desired goals or objectives of the project within the triple constraints of project management (William R. Duncan, 1996, p. 6). The main constraints faced during the completion of a project are scope, time, and resources.

The project manager is responsible to make sure that these constraints do not hinder the completion of the project. Additionally, none of these boundaries are to be violated in any form by any team member. For example, after the distribution of resources the team members are bound to work with the resources assigned to them. They cannot use the resources assigned to another team member.

Project management has become a crucial component of successful businesses all around the globe. Organizations are not only using project management for the introduction of their unique product or service, but they have synchronized their organizational goals with the successful completion of their projects and also to improve their internal operations and

handle the external threats. Many challenges arising from the competing business environment are tackled through project management by organizations on a daily basis.

Project management consists of four main processes that are applied to each project for it to be successfully completed in the given time frame (Metafuse, 2014). They are discussed below.

Initiation: The initiation phase of the project management process consists of defining the need or the main purpose of the project. It can be a business problem or creation of an innovative product. After the problem has been identified the solution is proposed and all possible solutions are evaluated according to their pros and cons.

From the problem to the formation of the final team, everything is documented at this phase. The project is finalized and the work is broken down into activities and assigned to the respective team member. A project manager is assigned to oversee the progress of the project. No actual work is performed at this stage, only the documentation of the project for kickoff.

Planning: In this phase, the project is planned and the solution is further scrutinized to look for possible errors. All the steps required to complete the project successfully are identified and the resources are allocated accordingly. Sequences are assigned to the tasks according to their significance and a strategy is formed for the project's completion. The cost of the project is also estimated at this phase under the supervision of the manager. The cost of the labor, resources, and outsourcing in some cases is calculated and a budget is prepared before starting the project. Therefore, the three fundamentals of the planning phase include task identification, schedule, and cost estimation otherwise known as scope, time, and resources.

Execution and Monitoring: During this phase the actual work on the project starts. Each team member works on their individual task. Their work is continuously monitored by the project manager in order to avoid human errors that could result in the project's failure.

The overall progress of the project is also monitored. The progress is then matched with the project plan and in case of failure aggressive actions can be taken by the project manager. Sometimes, the project manager also has the authority to fire a team member in the case of poor performance that cause delays in the project. Once a project deliverable has been completed, it is reviewed to check the quality and performance before handing it over to the client.

Everything is documented for the convenience of the team and the stakeholders of the project. The tasks are continuously monitored throughout the project and the project manager is in control with final responsibility for all decisions. In case of a problem, it is the responsibility of the project manager to act rationally and to come up with a feasible solution with the help of the team members. Additionally, the utilization of resources is monitored in order to avoid waste.

Close-Out: The final phase of the project management process is close-out. Here the final deliverables are handed over to the clients along with all supporting documentation. In the case of IT projects, or the like, such as software, supporting documentation would include such things as user manuals.

The team makes sure that regular updates, if needed, are delivered to the client and any problem at the client's end is handled carefully. At the end, the project is evaluated and analyzed for lessons learned. This analysis is helpful for the team in future developments of similar products. The team formed for the project is disbanded after the successful completion of the project.

The figure below clearly explains each phase of a typical project management life cycle. These processes are carried out for each project with the help of different project management life cycle methodologies. Project management life cycle methodologies are various ways that a project is planned, executed, and completed using multiple techniques and methods. Each methodology will be discussed individually later in this book.

Using these methodologies, each project is broken down into multiple tasks. These tasks are then completed individually or in a sequence depending on the project model selected for the project. In some cases these divided tasks are also the deliverables for the client and at times the final project is delivered all at once to the client without any periodic deliverables.

Project life cycles are used as roadmaps to determine the actions that must be taken to complete a project successfully. Since the project is divided into multiple tasks, the selection of critical activities and their sequence is easy to identify. The activities are then completed

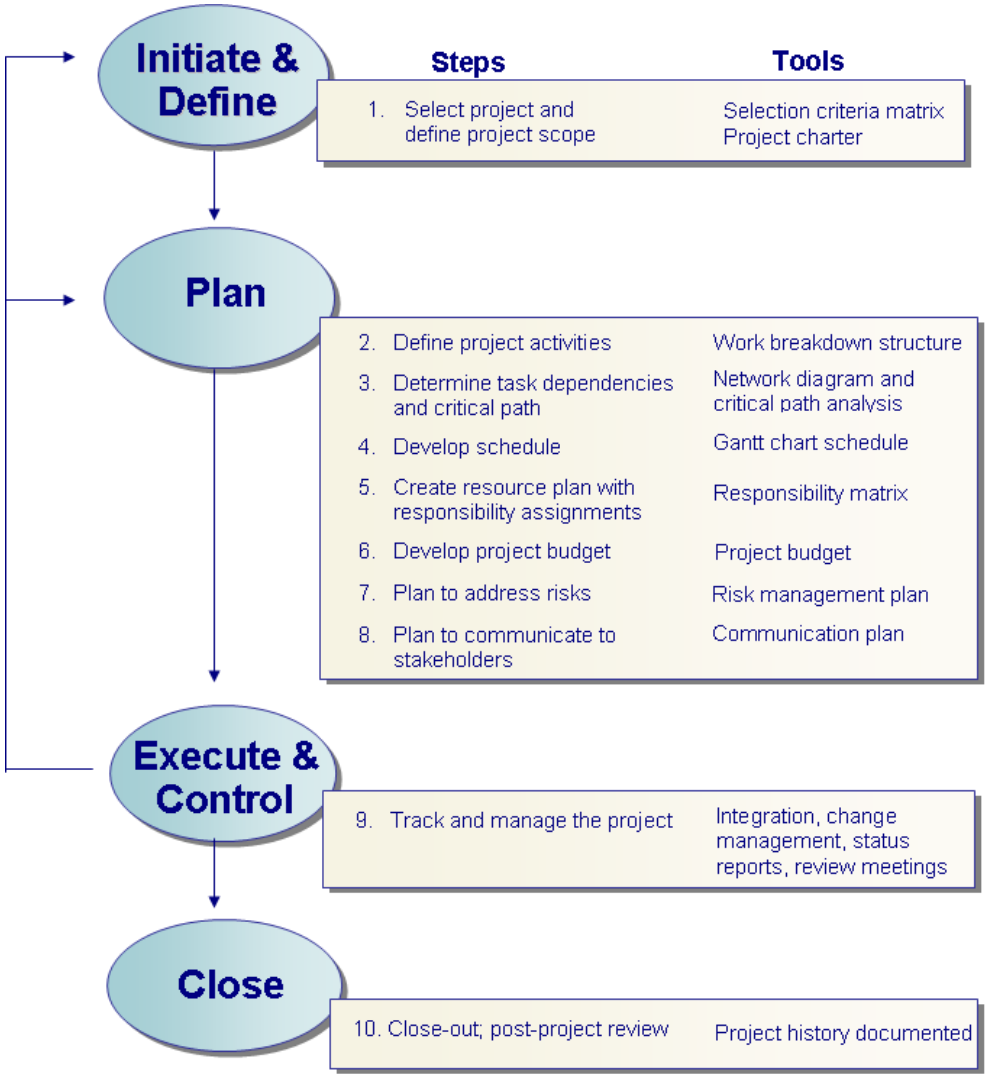


Figure I.1 Basic Project Management Life Cycle

according to their sequence. Any delay in the preceding activity will delay the overall project if the activity is on the critical path for the schedule. The project manager makes sure that each activity is completed on time to avoid project failure. Project life cycles usually drive the definition of two main criteria that are listed below.

- The technical work to do in each phase. The technical aspects of each phase are described in detail in order to avoid any confusion and eventual failure of the project.
- The person responsible to carry out the technical work. The person who will be carrying out the actual task is identified and assigned the task.

Once the project's details have been finalized the company can choose the best methodology suiting their requirements and execute their project accordingly. The following is a list of project management life cycle methodologies that a company might use.

Project Management Life Cycles are listed below.

- Rational Unified Process (RUP)
- PRINCE2
- System Development Life Cycle (SDLC)
- Capability Maturity Model (CMM)
- TenStep
- Agile: Extreme Programming (XP)
- Agile: Scrum
- Agile: Crystal
- Agile: Dynamic System Development Method (DSDM)
- Agile: Lean Development
- Agile: Feature Driven Development (FDD)
- Rapid Application Development (RAD)
- Unicycle Model
- Code-and-Fix Approach
- Scaled Agile Framework (SAFe)
- V-Methodology
- Waterfall Model
- Prototype Model
- Spiral Model
- Synchronize and Stabilize
- Reverse Engineering Development
- Structured System Analysis and Design Method (SSADM)
- PRAMIS
- Open Source

Project management life cycles are the foundation of project delivery. The selection of the method depends on the nature of the project. For example, for software based projects

a project manager may opt for using an Agile development model rather than another project management model because Agile development is better suited for software projects.

Though these models share some similarities among themselves, each has its own unique features that make them more suitable for certain projects. These methodologies are a complete set of guidelines needed for the successful completion of a project. From the start of the project till the end, all requirements and documentation details are explained clearly. The selection of the method is a critical decision as the wrong selection could lead to project failure hence affecting the image of the company negatively.

The methodologies provide a common process for team members to work in agreement with. Their main aim is to achieve the objectives and goals of the project in the given time frame and with efficiency. Using one methodology also makes the monitoring process easy, as it provides guidelines, framework, and documentation templates. In case something goes wrong, tracking will be easier since everything has been documented. The project manager as well as the team is provided with documentation for their assistance.

How to Use This Book

This book is not meant to be the definitive source on all aspects of project management or even the individual life cycles reviewed. Instead, it is meant to give a detailed overview of the project management life cycles that are available and a basis for choosing one. While many of these life cycles were developed for specific environments, they can be used in other unintended environments. However, that said, this doesn't mean that all project management life cycles can be used successfully in all project management environments.

Throughout this book you will see many references to software industry projects and practices. This is because many project management life cycles have been developed for the software industry or so that the reader will have a common theme to use for life cycle comparison. This does not mean that the particular life cycle can only be used for software projects or that it should be used for any industry other than software. It is up to the reader to understand the nature of their business environment and projects as well as the life cycles available and choose accordingly.

Each of the above mentioned methodologies are discussed in depth in individual chapters that follow. Their history, usage, and pros and cons are described in great length.

Before selecting any method for a project, it is crucial to go through them all to identify the model that is best suited for the project at hand to avoid failure and wasted resources.

Chapter 1

Rational Unified Process (RUP)

Rational Unified Process (RUP) is a software development process usually utilized by software companies. It is a comprehensive approach to software development combining the technical aspects as well as the documentation process. The main goal of RUP is the successful completion of the project by meeting the user requirements correctly and within the given time period.

History of Rational Unified Process (RUP)

Rational Unified Process has developed immensely over the years. It has improved greatly and many additions have been made to it that enhanced its productivity level. It is the result of collective efforts of many companies and people over the past few decades. Let us have a look at the rich heritage of RUP.

RUP came into existence in 1996 and was the creation of Rational Software Corporation (History of Unified Process, n.d.). It incorporates Unified Modeling Language (UML) and is widely known for its development tools. Rational Corporation based it on the Objectory Process written by Ivar Jacobson. It was originally developed by the Rational Software Corporation that IBM later acquired in February, 2003.

Since then many alterations have been made to RUP to aid customers with better development of their software. The Rational Unified Process includes the elements of project management, data engineering, configuration management and business modeling. They were added gradually after Rational Corporation merged with Pure-Atria in 1997 and the Requirements and Test Disciplines were added to the approach in the same year. Initially RUP was based on UML 1.0. In 1998, it was updated to the newer version of UML known as UML 1.1. The Business Modeling was added to the list of disciplines in 1998. Configuration and Change Management Disciplines were also added to keep up with the changing environment of technology (Kruchten, 2000).

In 1999 the Rational Unified Process was updated to UML 1.3 and Project Management Discipline was added to the process for real time development of software. This made the managing of projects much easier using RUP. It gained iterative development and

architecture from its Rational background and at this point all major elements were added to RUP. From 2000 onwards, only the tools to configure RUP and aid the customer with the installation and implementation of RUP successfully were incorporated (History of Unified Process, n.d.). Regular updates, techniques and tools are still added to ensure the efficient working of the process by IBM.

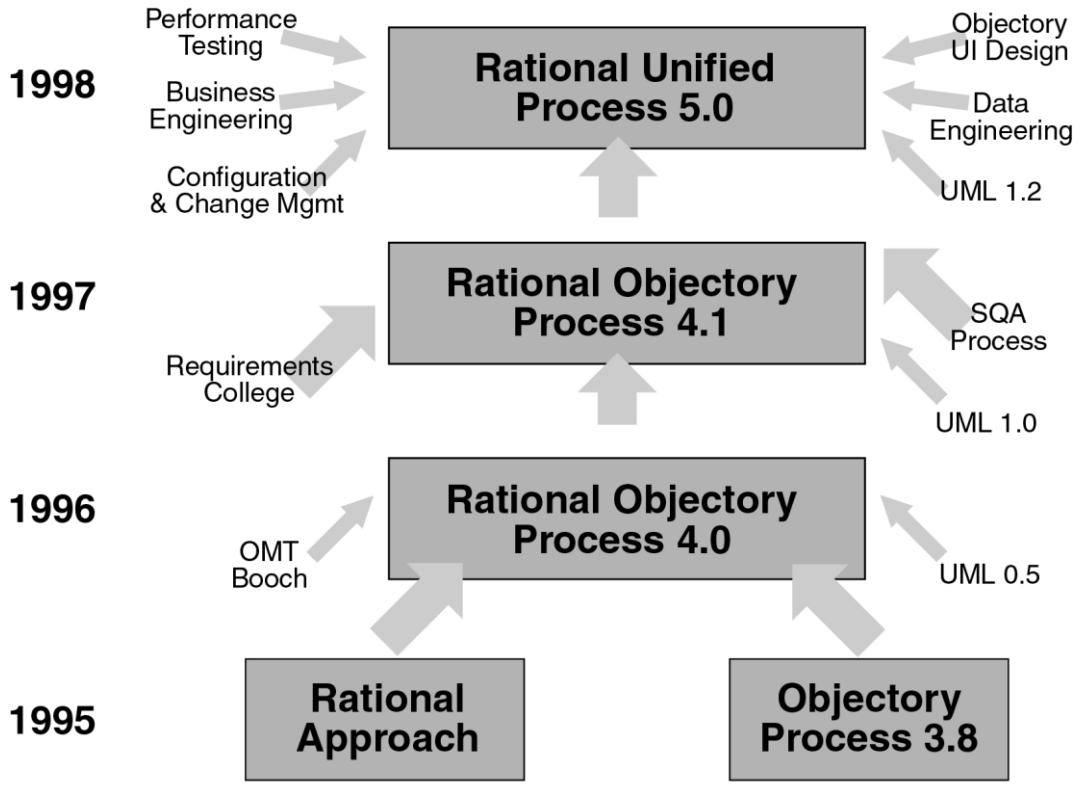


Figure 1.1 Rational Unified Process

Figure 1.1 clearly illustrates the gradual development of the Rational Unified Process over the course of decades (Kruchten, 2000).

This was a brief history of RUP and now let us cast a look at what exactly is Rational Unified Process.

What is the Rational Unified Process?

Rational Unified Process is a software development technique that provides guidelines and a disciplined approach for software development. It helps guide the software development process in a way that meets the customer requirements perfectly within the budget and time constraints of the project. It is an iterative mechanism and provides well-defined goals at the end of each phase (Paper, 2001).

RUP can also be referred to as a process product that has been developed and is maintained by IBM. They are working with their customers, business associates and other development authorities to continuously monitor and improve the experience of clients by improving the process. It provides a unified customizable framework to the client that can be molded according to the requirements of the software. Software developers can add or eliminate features in the process according to the requirements of their project.

The Rational Unified Process is basically supported by tools that automate the major parts of the process. Various models for development are created using these tools. They are also used for configuration process, testing of software components and for documentation of process details.

The Rational Unified Process provides clear and well-defined guidelines to all the team members. Each and every attainable goal is supported with knowledge-based guidelines and predefined methods and tools for successful completion of these activities. The advantage of sharing common knowledge with the entire team is that no matter what feature of the product is being developed first, all the members will be aware of the procedure, hence, avoiding mistakes and miscommunication.

They will also be aware of cost and resource allocations with the help of the Rational Unified Process Life Cycle. The Rational Unified Process is suitable for development of small projects as well as for highly complex and technical projects. Additionally, it has a Development Kit incorporated that provides configuration support in it to suit the needs of different organizations wanting to develop different kinds of software.

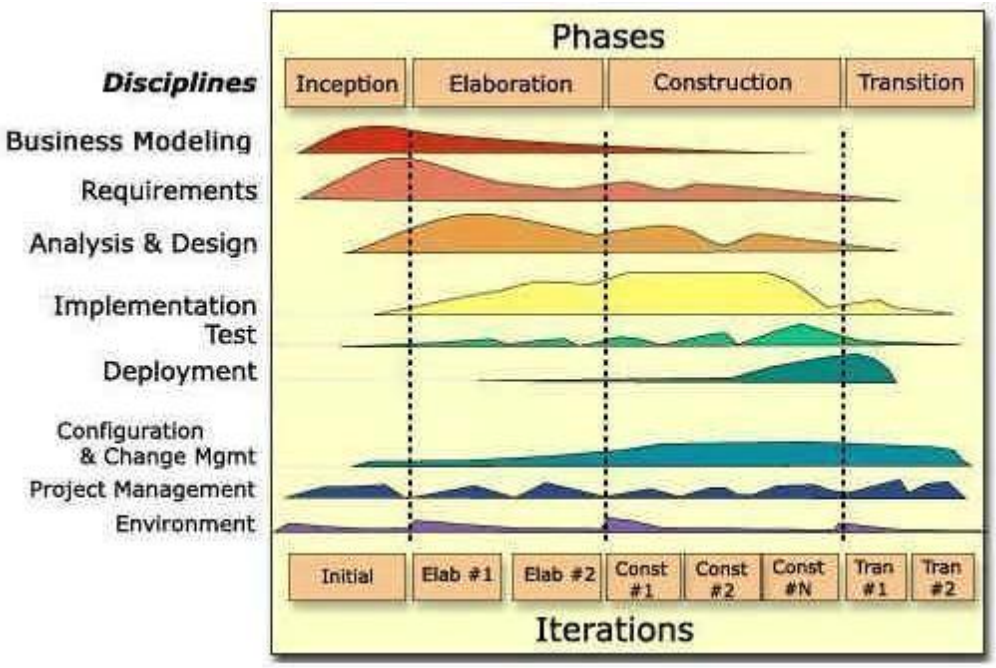


Figure 1.2 Alternative View of RUP Life Cycle

Figure 1.2 above illustrates the RUP life cycle. It is also known as the "hump chart" diagram (Ambler S. W., 2005). It is a two-dimension approach. The horizontal humps represent the role of each discipline in respective phases along with cost and time whereas the vertical axis represents the disciplines involved (Wesley, 2002). For example, Business Modeling is crucial at the inception phase and does not exist at the termination phase. Similarly, deployment is significant in construction and transition phases only. The diagram helps the team to understand the significance of each discipline at all four phases of the RUP life cycle.

The critical questions like *who*, *what*, *how* and *when* are always answered by an effective software development process. Likewise, RUP answers the questions below in order to run the process smoothly and flawlessly (Wesley, 2002).

- **Who:** Roles of each team member are defined explicitly.
- **What:** The artefacts or model on which the project will be based is developed.
- **How:** The performance of activities and their sequence is determined.
- **When:** The details of phases, iterative deliverables and sequence of the workflow are determined.

RUP is a combination of the best practices for software development techniques and tools. Opting to use it gives the developer an added advantage as most of the development plan is already formed and ready to execute. This saves time as well as money. RUP consists of the practices listed below (West, 2002).

- **Iterative Software Development:** Given the complex environment of today's development process it is often not possible to identify the problem, define a clear solution and then test the final result at the end. This will not produce the required result but will only waste time and scarce resources. To avoid such failure the development process is designed as an iterative process. The functionality of the system is developed in successive iterative components, where each component is part of the whole project (Paper, 2001). The selection of the requirements depends on the project risk. The most complicated and complex problem is addressed first to reduce the overall risk of the project.

- **Requirements Management:** A systematic approach is used to explicitly document all requirements before the start of the project. The incorporation of use cases and scenarios in the process has proven to be an effective approach to gather the functional requirements and to make sure that design of the software, implementation, and testing produces desired results for the end user. The changes made to any of the requirements are monitored and their effect on the overall process is noted down for convenience. It includes stating the requirements clearly in the requirements document and maintaining the traceability of these requirements in case of any changes made to them (Paper, 2001).

- **Component Based Architectures:** The structure of software is based on component architecture. It emphasizes designing an architectural approach that is easy to comprehend, adapts changes and is flexible. It reduces the complexity of a solution by dividing it into attainable goals and objectives. Each individual component represents a clear objective that fulfils a functionality of the whole process. This architecture is more vigorous and increases the probability of reusability (Brown, 1996).

- **Visual Model for Software Development:** RUP produces a clearly visual model indicating the tasks and their emphasis in significant graphical details. It hides

unwanted code structure but presents a vivid graphical representation of all the components and promotes a better understanding of the whole process by ensuring that the design and implementation aspects are on the same page (Grady Booch, 1999). Additionally, it avoids miscommunication as every single aspect is clearly illustrated in a modular form. Human errors can also be avoided due to this aspect.

- **Verifies Software Quality Continuously:** The quality of the development process is monitored from the beginning till the end. Rational Unified Process is embedded with Quality Testing that is executed at each phase of the development life cycle. Every iterative component is tested on the basis of its functionality and performance to look for faults before it is delivered to the client. Finding bugs and errors during the process is a lot less expensive than finding them at the end of the process (Paper, 2001).

- **Control Software Changes:** A disciplined approach is used to manage any changes that may occur during the development process. In software development, where changes are hard to accept due to the complex structure of development, RUP ensures that changes are accepted and implemented to enhance productivity. Changes may be made in the requirements document, resources, platforms, team members, technology, and more. It controls how these changes will be introduced and also when and who will be responsible for them. These changes are then synchronized with the functional components to avoid any delays. It also ensures that changes made to one component are isolated from other components, so that if they are not needed these changes will not interfere with their development process (Paper, 2001). If changes are not managed and communicated correctly to the team members then this will result in chaos among the team and will push the project towards failure.

These best practices are the result of the experiences of Rational's customers and the company itself. Furthermore, these same practices ensure a process that is capable of working in any given environment and fashion.

Rational Unified Process Phases

As mentioned earlier in this chapter the Rational Unified Process is a two dimensional process consisting of two different dimensions. The phases of the Rational Unified Process

fall under the time dimension (Ambler S. W., 2005). The process life cycle is divided into four crucial phases that are listed below.

- Inception Phase.
- Elaboration Phase.
- Construction Phase.
- Transition Phase.

Each phase has its own defined milestones that should be achieved within the given time and budget constraints. Milestones are used to mark the significance of an important event's occurrence. For example, the completion of the inception phase can be categorized as a major milestone. Figure 1.3 below explains the concept of milestones clearly.

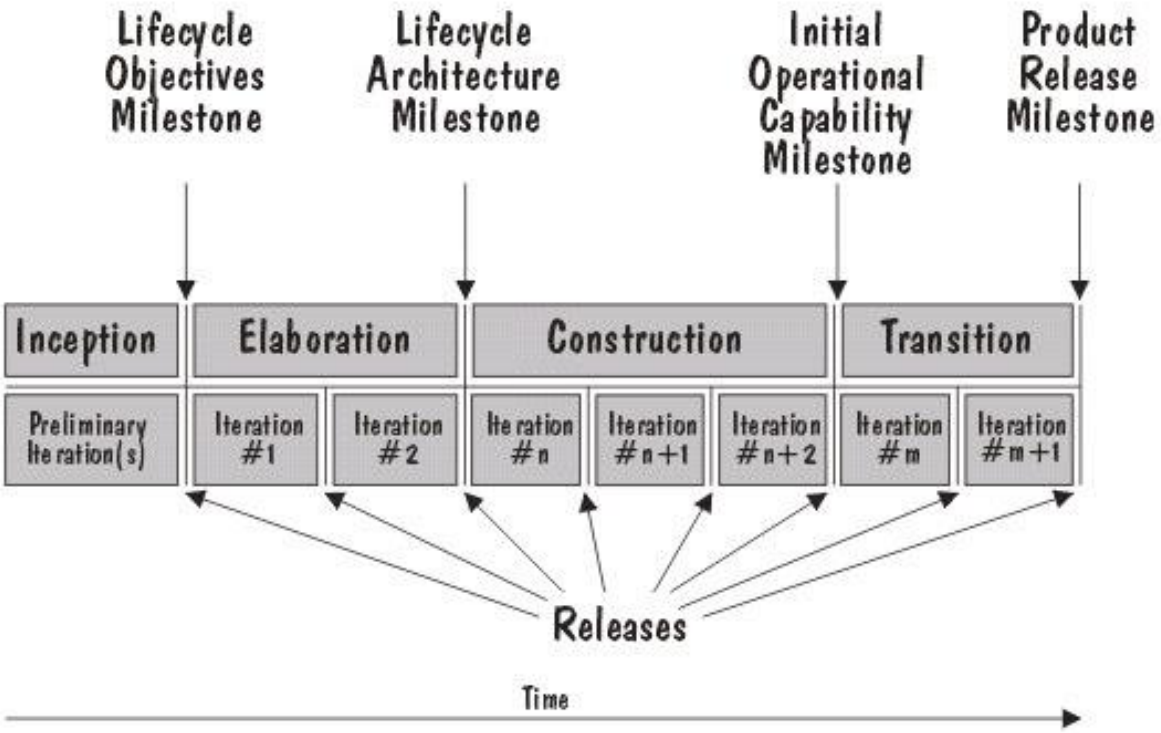


Figure 1.3 RUP Phases

Each phase serves a specific purpose and each objective is a milestone defined for the phase yielding some output (Ambler S. W., 2005).

Inception Phase

The inception phase is the first and the most important phase of the process. Here a clear vision of the project is defined with the help of the requirements obtained from the client. It is also referred to as defining the scope of the project. The primary purpose of the inception phase is to identify all the outside interactive sources (stakeholders) and gain their consensus regarding the project's outline and approve funding of the project.

A business case is defined including all the use cases and significant ones are elaborated. Furthermore, the business case is planned in collaboration with the client. After approval of the business case, the project plan, along with resource and budget estimations, are approved by all of the stakeholders. The project's feasibility and probability of success is determined at this phase by looking at the possible solutions proposed. The overall project plan is created and evaluated based on different scenarios. However, it is not an in-depth plan.

The basic functionalities and their development processes are included whereas their sub-division is left for the later phases. The business case consists of risk assessment, success criteria, estimation of resources needed and planning of phases with major milestones. The team members for the project are identified and assigned their respective responsibilities.

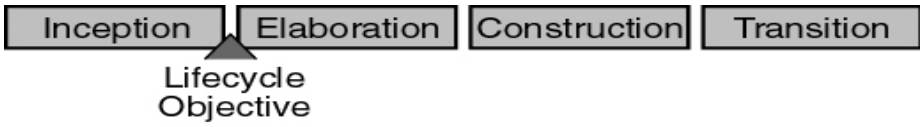
One or more than one architecture supervisors are assigned depending on the nature and complexity of project. Based on the activities of this phase a decision to proceed or not to proceed with the project is taken. If the project is approved, it indicates that the development team and stakeholders are mutually in agreement with the vision, project plan, risk assessment, and budget estimations.

The outputs of the inception phase are listed below (Paper, 2001).

- Vision document stating the vision, main requirements and functionalities and constraints.
- Initial use-case model which should be 10-20% complete.
- Initial project glossary including selected and critical features of the project.

- Initial business-case including context of the business, success criteria (market succession, client acceptance probability, future potential, etc.) and estimated budget.
- Initial risk assessment taking into consideration all possible risks involved with the development process.
- Project plan illustrating phases and possible iterations.
- Business model is required.
- One or more prototypes.

Milestone of Inception Phase: Life Cycle Objectives



At the end of the inception phase the first major milestone, Life Cycle Objective, is achieved. To evaluate the successful completion of the inception phase the following criteria below are applied (Paper, 2001).

- Stakeholder acceptance of scope definition.
- Stakeholder acceptance of success criteria.
- Requirements acceptance by the stakeholder indicating that they have been captured correctly and comprehensively.
- Stakeholder approval of the project schedule, risk assessment, budget forecast and development priorities.
- Forming of a risk mitigation strategy in accordance with all possible risks associated with the development process.
- Formation of the supporting environment as well as the acceptance of prototypes by the stakeholders.

In case the project fails to pass these milestones it can be cancelled and re-planned based on different speculations.

Elaboration Phase

During the elaboration phase, requirements are further detailed and the architectural design is finalized. The requirements are detailed to the point of making them easy to understand and comprehend the scope and the vision of the project along with the architectural risk associated with the development process (Ambler S. W., 2005). The purpose of the elaboration phase is to analyze the problem, develop a firm architectural plan, eliminate possible risks and form a project plan.

These objectives can only be achieved if the developers have a sound knowledge of requirements and the stakeholders are supporting the plan fully. The full knowledge of requirements is obtained from the Inception Phase. The architectural decisions, on the other hand, cannot be made without full evaluation of requirements, functional and non-functional components, and performance evaluation of the team.

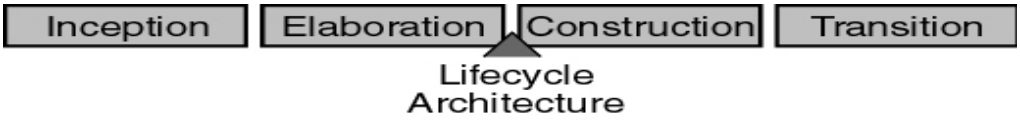
The elaboration phase is no doubt the most critical phase of the development life cycle. The major architectural design that will be the basis of the execution phase is finalized here. Any errors or faults in this phase can result in unsuccessful execution of the project, thereby resulting in project failure. The successful completion of this phase indicates that the most important requirements have been specifically identified and a clear plan has been formed to meet these requirements. A high-level project plan is also constructed indicating other phases and their major milestones (Paper, 2001). The crucial decisions related to use of technology and different methods for development are finalized in this phase along with the division of resources under the supervision of respective team leads.

At the end of this phase, hard and tough decisions have been made and the project is ready to start development. While the Rational Unified Process encourages changes, the elaboration phase ensures that requirements, architectural design, and the project plan are stable enough to execute the project without any major changes that may change the whole direction of the project. The budget plans are also finalized at this stage according to the project's plan. Construction of prototypes takes place in the elaboration phase as well. Prototypes are based on the scope, risk, size and concept of the project and they are made to aid the understanding of the development process for the stakeholders and also to mitigate risk.

The outcomes of the elaboration phase should be as detailed below (Paper, 2001).

- Use-case model should be at least 80% complete. All the actors and cases are fully identified at this phase along with the detailed description of tasks associated with them.
- Functional and non-functional requirements are captured and taken into consideration before starting work on the project.
- Descriptive Software Architectural Design.
- Architectural prototype that is executable.
- Revised risk assessment and business case.
- An overall plan for the development process including each iterative component along with evaluation criteria.
- Updated development case indicating the specific process to be used.
- A preliminary user manual. It is optional at this phase.

Elaboration Phase Milestone: Life Cycle Architecture



The second most important milestone is achieved at the end of the elaboration phase: Life Cycle Architecture. After the completion of the elaboration phase, its success rate is determined by evaluation of scope and objectives, the architectural design, and elimination of risk. The criteria below are used for evaluation (Paper, 2001).

- Stability of the vision.
- Stability of the architectural design.
- Identification and elimination of the major risk elements.
- Detailed and sufficiently accurate development plan with possible backup estimates.

- Mutual agreement of all stakeholders regarding attainability of the current vision.
- Actual expenditure plan approved by the stakeholders.

In the case that the project fails to meet any of the above mentioned criteria, it may be cancelled and an alternative solution may be proposed and evaluated.

Construction Phase

In the construction phase, actual work progress on the software beings. It is developed according to the development plan initiated in the previous two phases and each deliverable is thoroughly tested before delivering it to the client. The software project is developed within the time, budget and quality constraints and, like the construction of a house, in this phase the software is developed from beginning till end with iterative deliverables. The quality of the components is maintained and testing is done at each step to ensure that the quality remains intact. The main emphasis is to develop deployable software that satisfies the user requirements. Coding and testing is done at this phase of the development process. If deemed necessary, initial components are deployed earlier, either internally or externally, to gain customer feedback and rectify any mistakes made.

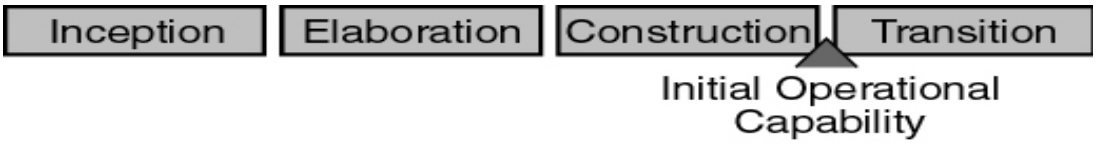
Usually, one iterative component is developed at a time but sometimes the project is large enough to initiate parallel development of a few components at the same time. This enables the developer to deliver more components to the client simultaneously and save time, but can also result in an increased level of complexity of the project and workforce manageability problems can arise. Correlation between the project plan and the architectural design is pretty high. In case one of them contains any ambiguity, the other is bound to fail as well. This is the reason why it is extremely important in the elaboration phase to develop a balanced plan and design.

The construction phase completes the whole software development life cycle along with the provision of necessary documents needed by the client to understand the project. These necessary documents include the instruction manual for the user. Hence, outcomes of the construction phase are listed below (Paper, 2001).

- Software created using the right platform.

- User manual.
- Current release description.

Construction Phase Milestone: Initial Operational Capability



At the end of the construction phase, the third major milestone, Initial Operational Capability, is achieved, software development is complete, and the product is ready to be deployed. The developers make sure that the software and the user are ready for operability and also the risk of failure is evaluated and at times fully eliminated. The first release is often known as the "Beta" release. The evaluation criteria for construction phase are listed below (Paper, 2001).

- Stability and maturity of the release.
- Stakeholder's acceptance of the software deliverable.
- Acceptance of the actual expenditures by the stakeholders.

In the case that the product fails to meet the expectations of the client, acceptance will delay the whole project by one release. At this stage, cancellation of the project is often not an option because resources and time have been allocated to the project and team members are fully engaged.

Transition Phase

The Transition phase is the last phase of the Rational Unified Process life cycle. Here the final version of the software product is deployed to the user and the testing process is conducted. User acceptance is the most critical point as it helps determine the success or the failure of the overall project. Once the product has been deployed, issues like minor faults, missing features, or postponed features drive the release of new versions of the product.

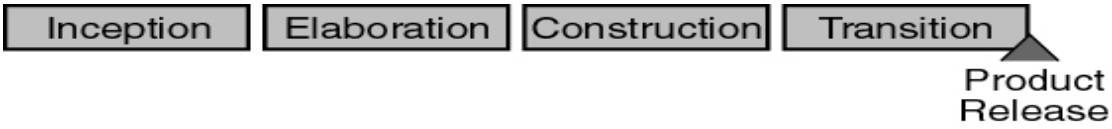
The software is deployed to the user at the end of the transition phase and it is complete only when the user accepts the product and it produces the desired results. Training of the end user and operational staff is also done at this phase and, for their guidance, instructional manuals are provided. No additional development takes place at this stage as all development processes should have been completed in the construction phase. By the end of this phase the project should be closer to its termination.

The transition phase should yield the outcomes below (Paper, 2001).

- "Beta testing" to determine whether the customer requirements have been fulfilled or not.
- Delivery of the software to its end user.
- Training of the end user and operational staff.
- Completion and delivery of the user manual.

The main focus of the transition phase is hand over of the product to the client. It usually includes more than one iterative release like, beta release, bug-fix release, general availability release and enhancement release. Extra efforts are invested in the completion of the user manual and training for the operational staff along with gaining feedback from the client. This phase can be very simple to highly complex depending on the nature of the product. For example, an update or a new version of an existing product release can be very simple whereas a completely new software product can be highly complicated.

Transition Phase Milestone: Product Release



At the end of the final phase, the fourth major milestone is achieved and it is Product Release. After its completion, the development team gains feedback through a post mortem of the project and evaluates the performance of the team as well as decides whether any further increments are required or not. The evaluation criteria for the transition phase are listed below (Paper, 2001).

- User satisfaction and feedback.

- Actual expenditures should not exceed planned expenditures.

The software development process comes to an end after the successful completion of the transition phase.

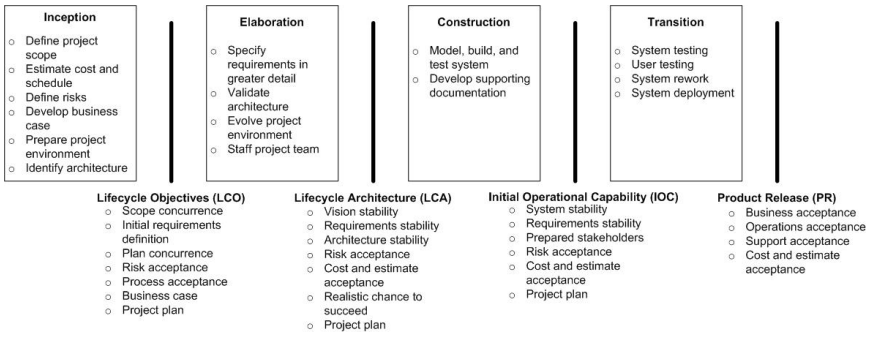


Figure 1.4 Summary of RUP Milestones

Figure 1.4 above summarizes the Rational Unified Process Development Life Cycle along with the four major milestones of each phase.

Advantages of using Rational Unified Process

Advantages of RUP are listed below (Najam Shahid).

- The Rational Unified Process is a complete methodology and its documentation is readily available.
- RUP is published, distributed, and supported openly.
- The Rational Unified Process is easy to comprehend. There are plenty of online walkthroughs and tutorials available that provide step-by-step guidance for the user. Some institutions also offer courses in RUP.
- RUP supports changing environments and minimizes risk.
- RUP's iterative nature divides the project into multiple tasks making coding and completion easy.
- High level of reusability of code.

Disadvantages of Rational Unified Process

Disadvantages of RUP are listed below (Najam Shahid).

- The process is highly complex. Unless the user is an expert or highly trained professional, the utilization of this process correctly can be problematic.
- Does not capture the sociological aspect of truly developing increments.
- May result in a totally disorganized form of software development.
- Software development integration sounds good on paper, but for highly complex projects having multiple development streams, it will only add to the confusion and will result in project failure.

The Rational Unified Process will not, by itself, guarantee success of the project. The developer has to make sure that all requirements, evaluations and estimations are captured and documented correctly. The choice of the development process highly depends on the nature of the software project. In a case of high complication, it would be best to avoid using Rational Unified Process.

Chapter 2

PRINCE2

PRINCE stands for Projects In Controlled Environment, Version 2. It is a controlled, measureable and flexible technique developed for project management which is suitable for all kinds of projects. It has been modified and updated regularly by professionals to cope up with the ever changing technical environment in today's businesses. It is available free of cost to users and is owned and controlled by The Office of Government Commerce (OGC), which is a very stable public authority and it is used by the UK government extensively in private sectors.

History of PRINCE2

PRINCE2 is a derivation from an earlier method known as PROMPTII and the project management based PRINCE method. The Central Computer and Telecommunications Agency (CCTA) developed it initially in 1989 to aid the development of IT based projects for the UK government. This shows that PRINCE2 was initially designed for IT based projects but apparently is now used outside this particular domain as well. Its extensive use and popularity led to the launch of a revised and modified version now known as PRINCE2 (i.e. Project In Controlled Environment, Version 2). It is a generic project management method released in 1996 and it is a de facto standard that is now used internationally as well. A consortium of 150 organizations within Europe contributed to the release of PRINCE2 (PRINCE2™ Project Management Methodology Overview and History, n.d.) .

After it was initially released in 1996, PRINCE2 was revised a couple of times to improve the performance further and provide users with an added incentive for use with a revised process for project development. It was revised in 2009 and a version named "PRINCE2: 2009 Refresh" was released. It was also referred to as PRINCE3, but the fundamental principals remained unchanged since 1996. Improvements and additions were only made to adapt to the changing business environment as well as to overcome weaknesses present in the previous method (Pincemaille, 2008).

PRINCE2 is a flexible process-based project development technique that is suitable for all kinds of projects due to its adaptive nature. Gone are the days when it was specifically

used for IT oriented projects only. Now, this project development process is being utilized in all forms of organizations throughout the globe.

What is PRINCE2?

Every organization needs to have the ability to adapt to changes. Sticking to a process that has been used for ages could result in a big loss for a company due to economic, environmental, and technological factors continuously changing. This does not mean that a company should abandon their previous practices altogether, but they should always have room for improvements and changes within their organizational structure as well as their production process. Changes ultimately are made through projects and PRINCE2 is an ideal project development process for any organization looking for changes within their production framework (Wideman, 2002).

Project In Controlled Environment, Version 2 is a project management methodology designed to cater to the needs of multiple kinds of organizations. It is flexible, easy to attain and understand, and produces the desired results efficiently and effectively. The essence of this method is that it effectively distinguishes between the techniques involved in the process and the management required to govern it. It focuses on the products that will be produced rather than the techniques used.

PRINCE2 has the ability to manage resources effectively in a controlled environment and handle the risks associated with the project. It incorporates the universally accepted and best practices of project management. Understanding the elements associated with PRINCE2 is quite easy and it enables the team to work in an understandable environment by minimizing the chances of miscommunication. PRINCE2 assumes that the team members working on a particular project have never worked in a similar environment before. Therefore, it provides a very well organized and explicitly defined development environment (PRINCE2 Project Methodology, n.d.).

Characteristics of PRINCE2 are listed below (Limited, 2013).

- **A Defined and Finite Life Cycle:** The development life cycle is clearly defined. The starting, middle and ending of the project's development process are marked and explained elaborately.

- **Measureable and Defined Business Products:** Each business product is defined clearly. They are measureable, and more importantly, their progress is monitored and measured according to the project's plan.

- **Achievement of Business Products with Corresponding Activities:** Activities required to produce each business product are clearly mapped out. The sequence of activities as well as the time required for their completion is calculated and included in the development plan.

- **Resource Management:** All the resources required for the development process are defined and managed effectively to avoid wastage. Resources are assigned to each activity before the production process starts.

- **Defined Responsibilities within an Organizational Structure:** Respective responsibilities of each team member are explicitly defined and assigned. This prevents any ambiguity about who is responsible for what and answerable to whom. A clear hierarchical structure is defined.

Below are listed the benefits to product development from the utilization of PRINCE2 for project management.

- An organized and controlled start, middle and end of the project.

- Regular project process review against business cases and the project plan.

- Flexibility regarding decision points.

- Any deviation from the project plan is automatically controlled and rectified.

- Involvement of stakeholders and management as needed.

- Excellent communication between the project management team and the rest of the organizational units.

- Agreement regarding the initial requirements and continuous monitoring of the progress with respect to these requirements.

The PRINCE2 process is based on the concept of seven themes, seven principles and seven processes. Each of them will be discussed further in this chapter. Let us have a look at the seven principles first.

Seven Principles of PRINCE2

PRINCE2 is designed in a way to include all the possible aspects associated with the development of a product. Therefore it has been divided into 3 categories, including, principles, themes and, finally, process.

The principles are used as the guidelines to initiate a project plan that is both relevant to the design process and is helpful to the development and project management team. They can also be described as a list of common sense values put together for guidance of the team. Additionally, they are easy to apply and comprehend.

Being a principle-based project provides the added advantage of being able to adjust to any size, kind, and type of project. These principles can be applied universally on both small and large sized projects. They have been proven to be the best applied principles over the years and they are known to achieve required results and produce the best possible outcomes. Applying these principles gives the team members an advantage of being aware of the situation and it gives them a sense of empowerment because they have a clear road map to follow in order to reach their goals. The seven principles are listed below (Principles & Method, 2009).

- Business Justification.
- Learning from Experience.
- Defined Roles and Responsibilities.
- Manage by Stages.
- Manage by Exception.
- Focus on Products.
- Tailored to Suit the Project Environment.

Below the principles are discussed in detail.

Business Justification

The first and foremost principle is known as business justification. There should be a clear justifiable reason as to why the project is started in the first place. The reasons are documented to avoid variations during the development process. Every action taken during the project should justify the main goals and objectives of the project. The emphasis is on the importance of having a clear and detailed business case. The business case is updated and changed throughout the production process and in the case that there are some factors that do not align with the project's objective then they may result in the premature termination of the project (Buehring, 2013).

The business case includes the logical and valid reasons for running the project and it is then documented in measureable terms. PRINCE2 believes that a project is subject to tolerances and in the case that a project falls below a tolerance level, the business case will no longer be valid.

For example, if the project is started with the intention of earning a profit, then the business case determines the factors that will result in making a profit. In the case that the project is not anticipated to be profitable, then it should be stopped immediately (Buehring, 2013).

The PRINCE2 project development process breaks down a whole project into multiple stages and, at the completion of any stage, the business case is updated and approved by the management team for the project. The project's management team is responsible for the assessment of the project boundaries and its approval. At this point, the management team decides whether the business case is reliable for the project plan or not. In case any problems are identified, they should be brought to the attention of the project's management team before the stage assessment is completed.

In case the project is closed prematurely, the team should document any lessons learned for future reference and maximum effort should be made to benefit from the outputs produced by the project.

Learning from Experience

Human beings are prone to repeat their mistakes. However, repeating one mistake over and over again can be tedious and will always result in failure. In order to avoid this

situation, PRINCE2 encourages the user to learn from past experiences. The PRINCE2 methodology makes it compulsory for developers to report all lessons learned during project execution and, throughout the project, it is the duty of the project manager to remain alert and document any significant events. Lessons learned should be recorded at the end of each stage of the project. The lessons learned report is then sent to the management team for their review and approval. The management team also distributes the copies of the reports to various stakeholders throughout the organization. Due to this, the people within the organization will be able to avoid making the same mistakes made by the development team of this particular project (Buehring, 2013).

PRINCE2 also encourages the use of lessons learned from previous projects. During the kickoff of any project, the project manager is required to maintain a lessons learned log that contains previously learned lessons as well as the new ones learned during the project. Sometimes a person who has worked on a similar project is hired again to guide the team effectively to avoid repeating previous mistakes (The 7 PRINCE2® Principles: A Closer Look, 2010).

Defined Roles and Responsibilities

Defining roles and responsibilities makes it easier for each person involved to understand what he or she is expected to achieve. When there is a clear hierarchical plan defined and duties assigned, the chances of making mistakes is reduced and each person is more likely to focus on completing their task efficiently rather than being confused about what exactly is expected of them. This principle defines a three level hierarchical plan for the project (Buehring, 2013).

- **Management Levels:** A clearly defined management hierarchy should be present before starting the project. The project steering committee is at the top most level with the project manager next in line and team managers at the lower level. This ensures that the project is governed using three different levels of management.

- **Project Board:** Every PRINCE project must have a board consisting of an executive, senior user and senior supplier. There will always be just one executive. There can, however, be multiple senior users and suppliers. The executive ultimately makes all the important decisions related to the project with the help of the board members. For small projects, all the board duties may be

performed by the executive alone. For larger projects, the board's structure is more complex consisting of cross-functional members from different departments within the company.

- **Stakeholders:** Three main stakeholder's interests are identified by PRINCE2. The business, supplier and user interest. The project board is responsible to protect the interest of each one of them and the business interest makes sure that the money invested in the project will not be wasted. The supplier interest represents the provision of expertise and resources and the team may be recruited internally or externally. The user is generally interested in whether the project will be fulfilled by successful completion or not.

Manage by Stages

PRINCE2 promotes the concept of working in stages. There must be at least two stages: the project initiation stage and the delivery stage. Each stage should be clearly defined, planned and executed to fulfil the requirements of the customer. The initiation stage makes sure that the project is planned according to the time, budget, and quality constraints. This stage should not be skipped at any cost as planning is very important for the successful completion of the project. At the end of each stage, the achievements and documents are reviewed by the project board (Buehring, 2013).

Manage by Exception

Tolerance levels are set related to each stage while using PRINCE2. They establish project limits and boundaries. When the project exceeds tolerance levels, corrective actions are taken by management to get the project back on track and the project board is provided with regular updates related to the project's progress. This makes the monitoring and decision making process easier for them (The 7 PRINCE2® Principles: A Closer Look, 2010).

The tolerance level is set for the 6 performance targets listed below.

- Time.
- Cost.
- Quality.
- Scope.

- Benefit.
- Risk.

By managing the project this way, senior management does not interfere with the team's work unless some important decisions need to be made. This leaves senior management free to work on other pressing issues (Buehring, 2013).

Focus on Products

PRINCE2 recognizes the importance of product delivery, hence, focus on product principle is used. If the delivered product meets the customer requirements only then will the project be successful, therefore the product should pass all quality tests before being delivered to the client. In order to meet the customer requirements successfully, everyone on the project team should be aware of the requirements or there could be confusion among the team members and the product will not be developed according to the customer's needs (Buehring, 2013).

Tailored to Suit the Project Environment

The word 'tailored' here depicts the ability of PRINCE2 to mold itself according to the nature of the project. Its adaptive nature has made it one of the most famous project development processes around the globe. PRINCE2 tailors itself according to the project's culture, location, risk, requirements and resources available. This makes the work of the development team easier as the PRINCE2 process has the ability to adjust itself accordingly (Buehring, 2013).

Seven Themes of PRINCE2

The seven themes of PRINCE2 are the issues that should be addressed before starting the project. They are listed below (Service, 2007).

- Business Case.
- Organization.
- Quality.
- Risk.

- Planning.
- Change.
- Progress.

Business Case

Business case is the most important aspect of PRINCE2 as it determines the desirability of the project. Additionally, it determines whether the project is viable enough to be continued or should be stopped (Kerr, n.d.). The business case is used, updated, and reviewed throughout the project development process (Service, 2007).

Organization

The purpose of this theme is to determine who answers to whom in the project hierarchy. It clarifies the responsibilities of each team member.

Quality

The main purpose of this theme is to make sure the product passes all quality tests. The quality of the product should be maintained throughout the project.

Risk

The risk theme analyzes all possible risks associated with the project and comes up with ways to minimize or eliminate them. The risk evaluation is done at the end of each stage.

Planning

This theme is used to determine the means of product delivery and who will be responsible for each task.

Change

Its objective of this theme is to identify and control any possible changes that might occur throughout the development process.

Progress

Progress is, of course, a very important theme of PRINCE2. The objective is to track the development process and evaluate the achievements. This is also used to forecast the future performance of the project (Service, 2007).

Seven Processes of PRINCE2

The process framework of PRINCE2 is based on the principle of common sense. There are seven major aspects of this project management life cycle that every project opting for PRINCE2 should address to some extent. The process framework is tailored in a way to suit the needs of individual projects according to their nature. Completing a project using the PRINCE2 project development process is pretty easy as the process does not involve any complicated technicalities. The process is rather simple to understand and comprehend. The seven aspects of PRINCE2 are listed below (PRINCE2 Methodology, 2008).

- Directing a Project (DP).
- Starting up the Project (SU).
- Initiating a Project (IP).
- Controlling a Stage (CS).
- Managing Product Delivery (MP).
- Managing Stage Boundaries (SB).
- Closing a Project (CP).

Each of these stages of the development process will now be discussed individually. The diagram below clearly explains each stage of the model.

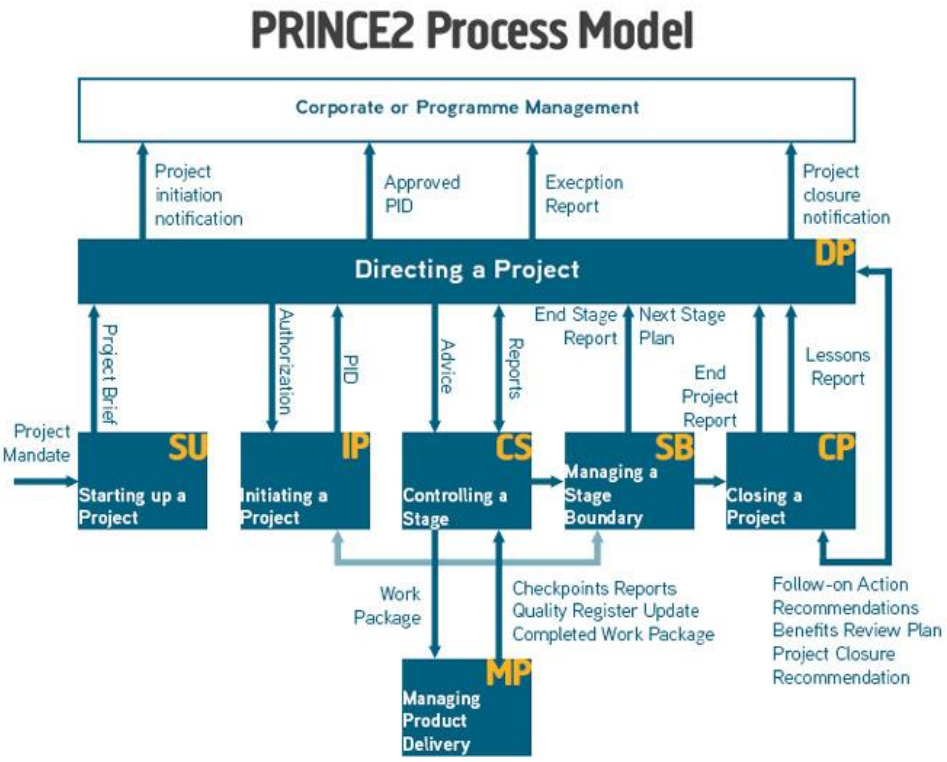


Figure 2.1 PRINCE2 Process Model

Directing a Project (DP)

The process describes in great detail how the governing authority, usually the supervisor of the project management team, should direct the process. The responsibilities are highlighted along with their respective designations. The people included in the directing process are suppliers, supervisors, and end-users. Their agreement is essential on some major aspects of the project.

The directing process covers the necessary steps that are required for the successful completion of the project and they are mapped out from the beginning of the project until the very end. This reduces the chances of miscommunication and chaos as every person will be aware of their responsibilities. It is usually aimed at the process board and it also describes how the project board should react in case of an unforeseen problem or hindrance and how it should be tackled effectively. This also gives the board the right to close down any activity that they feel is not benefiting the development process. The five major steps of this activity are listed below (PRINCE2 Methodology, 2008).

- Authorization of the business cases and the project plan for the development process.

- Approval of the project's goals and objectives.
- Realistic and justifiable key points in the project.
- Monitoring of the whole process and rendering any advice, if deemed necessary.
- Controlling the closure of the project, making sure that it ends on time and meets all the given requirements.

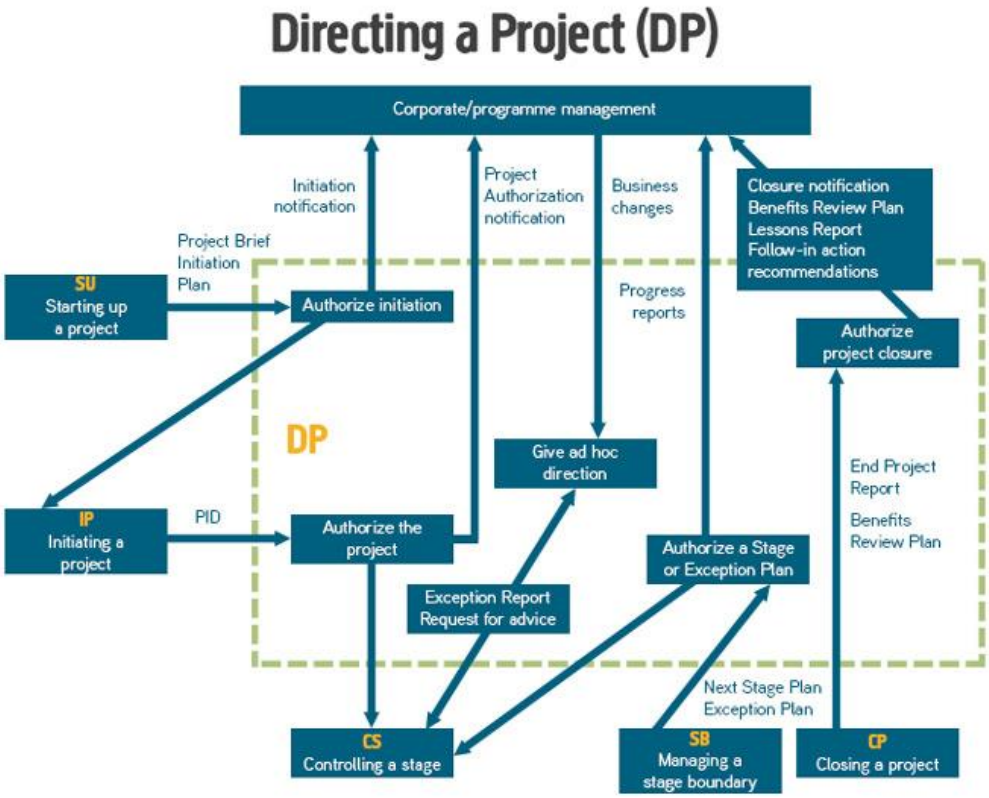


Figure 2.2 Directing a Project (DP)

Starting up the Project (SU)

This stage involves the pre-processing of the project and the project plans and the team are finalized at this step. All the necessary information is gathered and documented to use throughout the development process. The team members are asked to work on the initial project plan and it is approved by the project board. After its approval the work on the next stage is initiated. It is a very short pre-process with six important objectives that are listed below (PRINCE2 Processes, n.d.).

- Knowledge of the project's fundamental objectives.

- Appointment of the project management team and finalization of the initial design.
- Decision regarding the approach that will be used to form the solution to the problem.
- Identification of the customer quality expectations.
- Identification of the project tolerance determined by the higher level of management within the organization.
- Planning of the work and drawing of the PRINCE2 contract between the team and the customer.

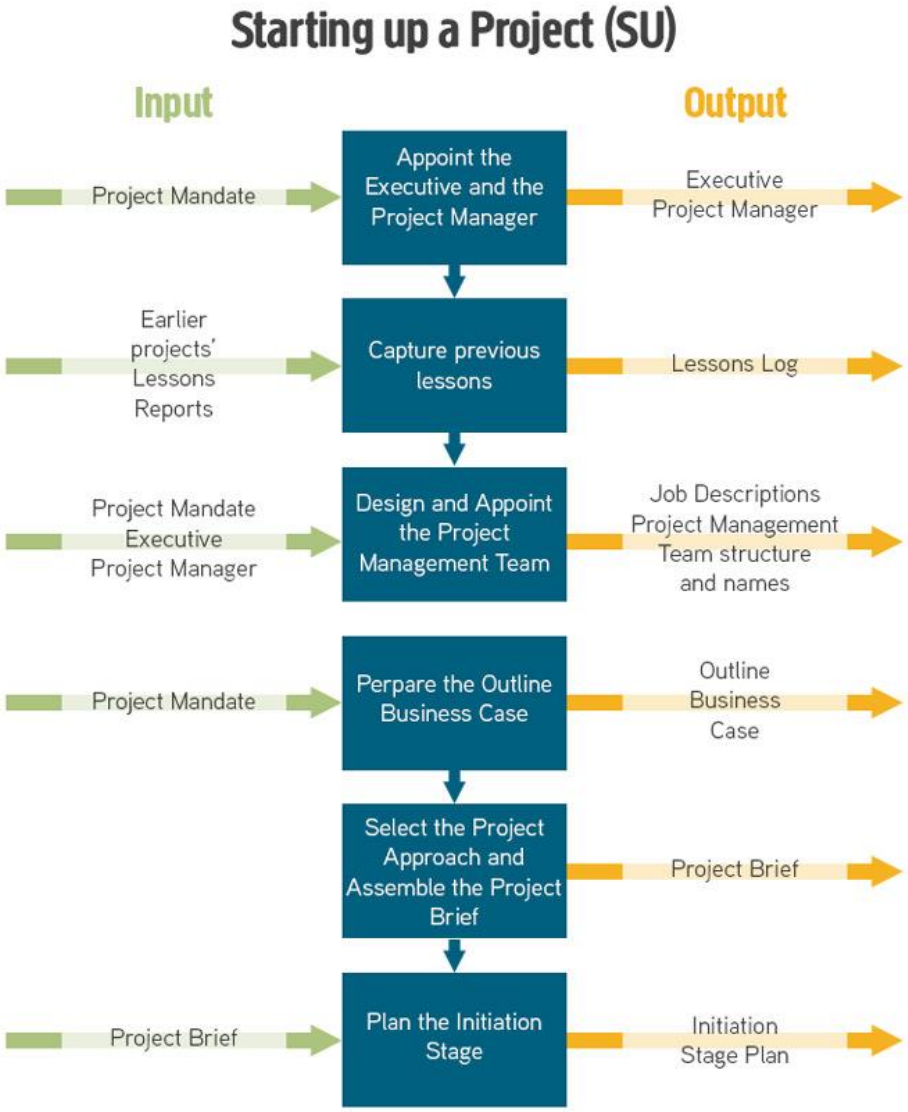


Figure 2.3 Starting up a Project (SU)

Initiating a Project (IP)

This phase makes sure that all the requirements documented are realistic and the project plan is justifiable and achievable. The start-up plan for the project development process is initiated and project files and necessary documentation are created to govern the development process smoothly. Decisions like whether the project has the probability of being successful or not, are enough resources available, and is the client in agreement with the project plan are discussed and agreed upon at this stage. Initial documentation is produced in this phase and success rate is determined. If the project does not look successful in the future then it is either terminated or re-planned. Key activities in this phase are listed below (PRINCE2 Processes, n.d.).

- Planning and maintaining quality throughout the development process.
- Planning the whole project structure and which techniques to use.
- Revision of the project risk and business cases.
- Project controls are initiated.
- Project documentation and files are created.
- Assembling of the Project Initiation Document is completed.

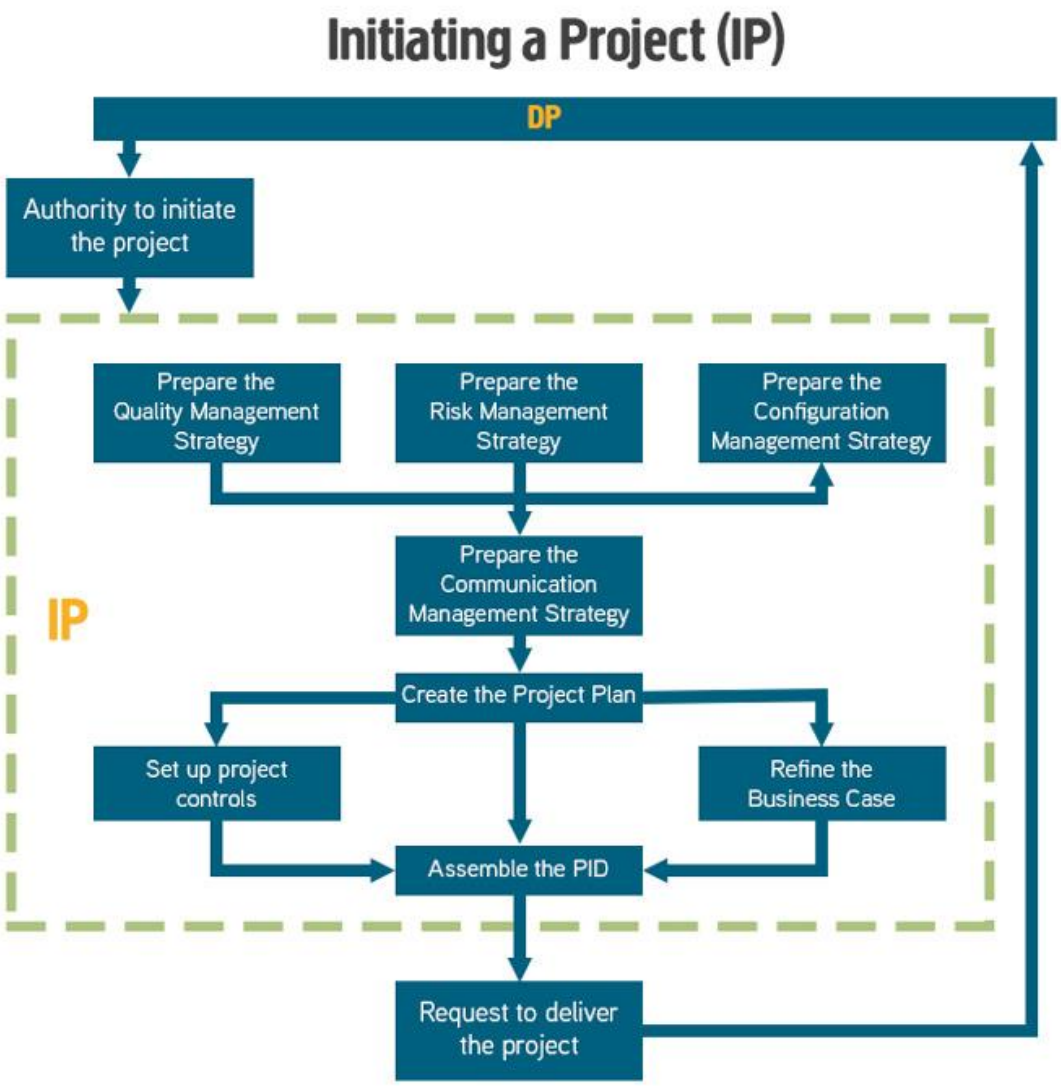


Figure 2.4 Initiating a Project (IP)

Controlling a Stage (CS)

This process describes the controlling activities that need to be undertaken by the project manager. It is the responsibility of the project manager to ensure that the project stays on track and is not affected by any unforeseen activity and, in the case it does happen, precautions should be taken by the project manager. Each activity performed by the project development team is controlled and monitored by the project manager. The day-to-day activities are monitored and any changes that may occur are monitored and controlled. These activities are carried out along with the ongoing activities of Risk Management and Control. The controlling stage includes the areas below (PRINCE2 Processes, n.d.).

- Authorization of the work needed to be done.

- Progress information related to the work is gathered.
- Monitoring changes.
- Monitoring alterations in risk.
- Revision of plans.
- Reporting to executives.
- Taking actions deemed necessary according to the situation.

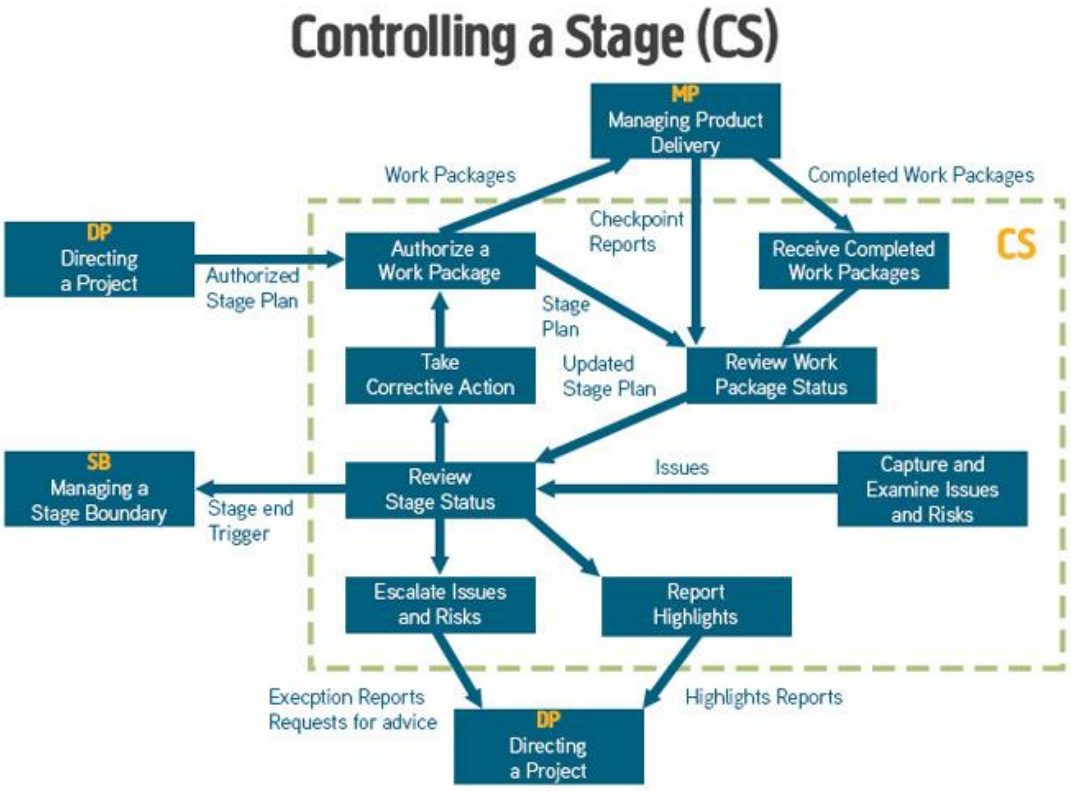


Figure 2.5 Controlling a Stage (CS)

Managing Product Delivery (MP)

The teams involved in the development process need to work together in order to deliver the product on time to the client. This process provides a control mechanism that ensures timely delivery of the products that pass quality standards. The work agreement between the Project Manager and the Team Manager is called a work package. It includes target dates, reporting requirements, and quality metrics. The managing product delivery (MP) process includes the areas below (PRINCE2 Processes, n.d.).

- Allocation of the work to the team is authorized.
- Planning of the team work.
- Making sure that the work is done.
- Making sure that the product passes quality tests.
- Reporting to the Project Manager regarding the development progress and quality.
- Acceptance of the finished outcome from the client.

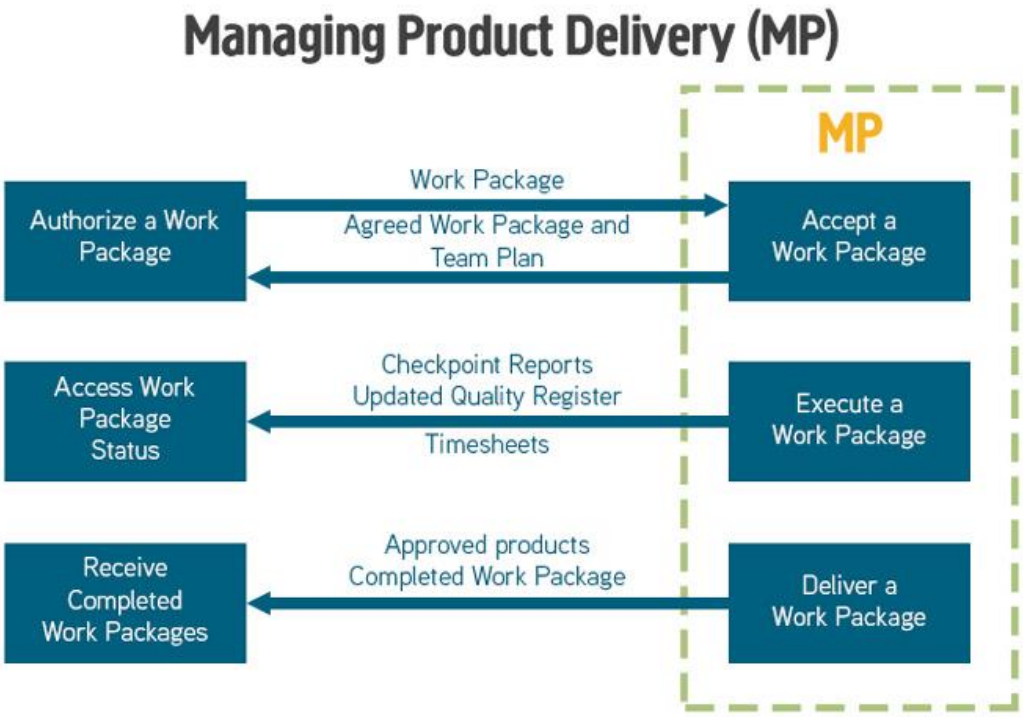


Figure 2.6 Managing Product Delivery (MP)

Managing Stage Boundaries (MB)

Controlling a stage process indicates what should be done at the beginning and during the stage whereas managing the stage boundaries indicates what should be done at the end of the stage. The aim is to see whether the project should be continued or terminated. If it is to be continued further then the planning of the next stage is initiated, the business case is modified, the project plan is expanded and a future risk assessment is done. Deciding what is to be done about a stage that has crossed its tolerance level is also covered here. At the end, the completed stages are reported to executive management. Managing stage boundaries includes the areas below (PRINCE2 Processes, n.d.).

- Making sure that all work is finished from the current stage.
- Planning of the next stage.
- Project plan is updated.
- Business case is updated.
- Risk assessment is also updated.
- A detailed report of the stage just completed is produced. It should include the performance and the outcome.
- Approval of the project board is sought to move forward with the project.

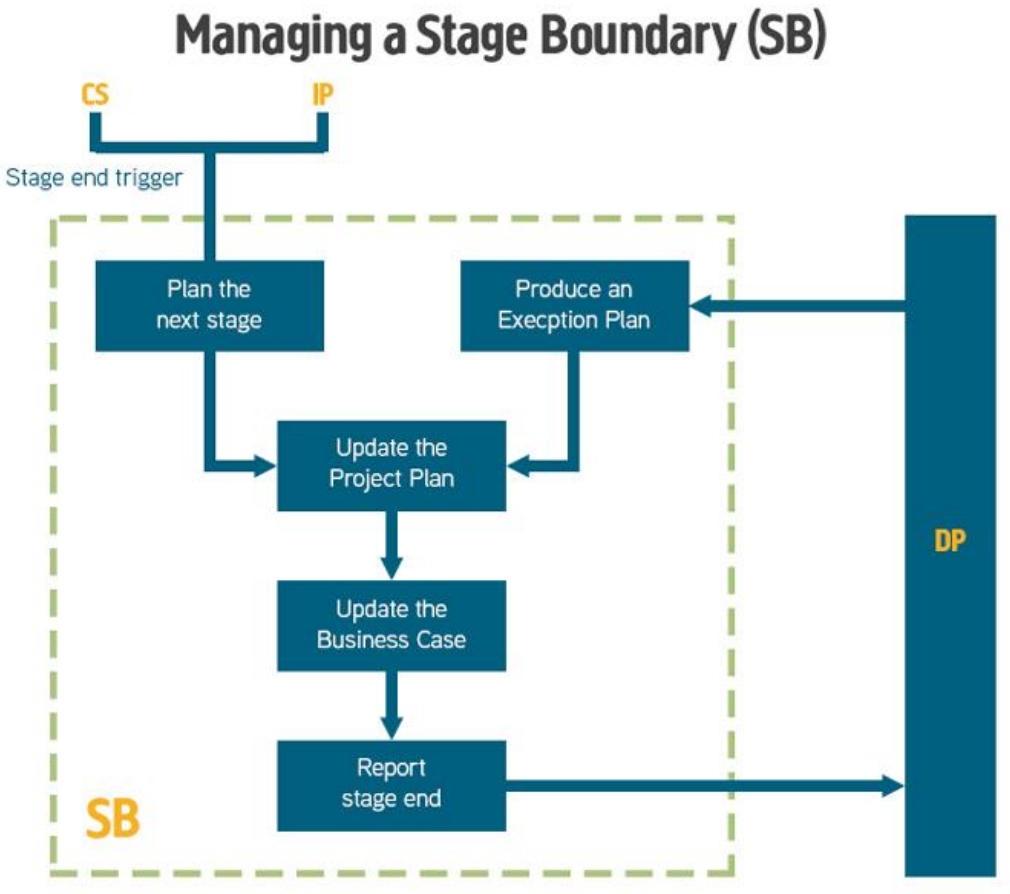


Figure 2.7 Managing a Stage Boundary (SB)

Closing a Project (CP)

This process covers the closure of the project. It is either its natural end or it can be terminated early due to troubles and problems. The team manager needs the approval of the

team before he/she goes further with the decision. Before the project is closed, delivery of all customer requirements is verified. The teams formed for the project development process are terminated and the resources are freed up. CP includes the following aspects below (PRINCE2 Processes, n.d.).

- Analysis of the extent to which the initially stated objectives have been met.
- Customer's satisfaction analysis through feedback.
- Maintenance and support arrangements.
- Recommendations for any follow-up actions needed.
- Work analysis of the team and any new lessons learned are documented for future reference.
- Reporting on the project management's activities and determining whether they were successful or not.
- Development of a plan to check the achievements of the product in the market and its future potential.

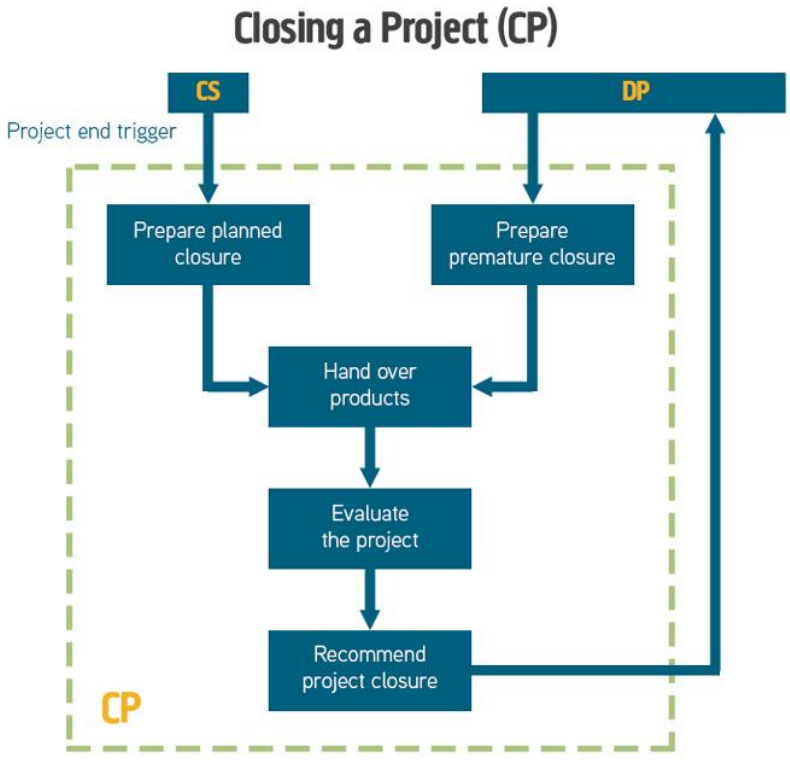


Figure 2.8 Closing a Project (CP)

Advantages of PRINCE2

Listed below are advantages of PRINCE2.

- Product-based project life cycle that divides the process into stages making the management task easier.
- Enhanced communication among the team members as well as the stakeholders.
- Stakeholders get to have a say in decision making.
- It encourages improvements within an organization by encouraging the members to learn from their experiences.
- It is flexible and adaptive in nature which is tailored to suit the needs of any project requirements.
- It is designed to handle and deal effectively with changes.
- Using PRINCE2 helps improve customer satisfaction.

Disadvantages of PRINCE2

Listed below are disadvantages of PRINCE2.

- Everything, whether formal or informal, needs to be approved by the project board.
- Each stage, phase and component of the process must be defined in great detail. Also, every detail should be documented. This makes the whole process rather tedious.
- In case the project crosses a tolerance level, it may be shut down resulting in loss of resources and time.
- Application to small projects is rather difficult.

PRINCE2 has gained popularity due to producing desirable results over many years. It has been improved greatly to provide customer satisfaction. However, the choice of a project development life cycle depends greatly on the nature and size of the project (Pincemaille, 2008).

Chapter 3

System Development Life Cycle (SDLC)

The System Development Life Cycle, commonly known as SDLC is a software development process designed specifically for the IT environment. It provides a detailed description of development rules and regulations with respect to a project's goals and objectives. Additionally, it ensures that all user requirements are fulfilled and the software consists of all the required functionalities. Furthermore, it defines a clear structure for the development process and the project is developed on the bases of this structure. Before we go into the details of SDLC let us have a look at its rich history.

History of System Development Life Cycle (SDLC)

The System Development Life Cycle came into existence in the 1960s. It was specifically created to handle large scale business development projects at that time. Since the business development industry was booming, they needed a life cycle to manage the project process effectively. It is a very methodical process with each stage clearly and elaborately defined.

Several mission critical projects have been managed using this process. For example, Structured System Analysis and Design Method (SSADM), was produced for the UK's Office of Government Commerce. Since then, constant changes and improvements have been made in this process to cater to the needs of the users desiring the production of quality software (System Development Life Cycle, 2010).

Daniels and Yeates defined this project development methodology completely in 1971. The System Development Life Cycle was clearly designed for IT based projects. The management of a project of any other nature using this process might result in failure.

What is SDLC?

SDLC is a clear defined plan for software development. It is a process that is divided into stages to make the development process easy and achievable. It is widely used in software engineering, system engineering, and information systems to develop unique and innovative software.

Like all other project development processes, SDLC also has a clearly defined plan that needs to be followed by the development team. The road map helps in defining each component of the process clearly eliminating the chances of ambiguity among the team. Each stage of the process life cycle is elaborated explicitly. Software developed using SDLC will ensure that the customer is satisfied and that the project is completed within the triple constraints of scope, time, and resources.

The purpose of SDLC is to provide the developer with a process that ensures successful implementation of software by meeting or surpassing customer requirements and business objectives. The details of the process are documented and distributed among the team members to ensure smooth operations during the process as each member will know what, when, and how to perform their duties. Additionally, it provides the project manager with clear guidelines to manage and monitor the project.

SDLC incorporates the use of Records Management (RM) and this management is established in the early stages of the SDLC process. Each important aspect of the process is documented and a checklist is formed to cross check the functionalities. The checklist can also be used for other development processes.

The checklist contains three to five important questions related to each phase of the System Development Life Cycle (SDLC). These questions are intended for the recordkeeping and management of the project. The questions are then discussed with the stakeholders to get their opinion and approval regarding certain aspects. The checklist can be modified by the organization to suit their needs for the development process. For example, some organizations utilize a five-step SDLC process while others use a ten-step SDLC process. These organizations will alter the checklist accordingly.

Below is a template of a SDLC checklist.

SDLC Goals

The goals of the System Development Life Cycle are listed below.

- To deliver a quality product that fulfils all customer requirements within the assigned budget and time.
- Delivery of a measurable and repetitive process that is designed to develop quality software.

- Development of a project management process to ensure that each stage of the process is continuously monitored and managed by the manager effectively.
- Identification and assignment of designations and responsibilities to all the team members according to their abilities and skills. Technical and functional managers should be aware of their responsibilities and should work effectively.
- The requirements of the client should be defined carefully and clearly for the team to satisfy them.

SDLC Objectives

The above mentioned goals must be achieved. SDLC assists in the achievement of these goals by use of the methods below.

- Establishing management levels defining responsibilities, assigning authorities, establishment of direction, providing management control, and continuous review and approval of the process by the stakeholders.
- Ensuring accountability of project management.
- Documenting requirements at the beginning of the project and tracking throughout the development process.
- Ensuring the project is completed using the assigned resources within the given time.
- Identification and management of project risk.

System Development Life Cycle Phases

SDLC is designed to alter itself according to the requirements of the project. It has ten phases but not all of them are applicable for every project. They may overlap each other or may not be included at all. The selection of which phase to include depends on the requirements of the client and the nature of the software.

During these phases the documentation, work plan, and work products are defined and approved. A clear software development plan is created with respect to the requirements. The team members are then assigned their respective tasks and the work begins. Each phase has a

deliverable that should pass quality checks before deployment to the client. The final phase of SDLC occurs when the project is completed successfully or comes to an abrupt end due to some reason. The primary object of SDLC is to provide a quality product using ten pre-defined phases with clear objectives.

The ten phases are list below.

- Initiation Phase.
- System Concept Development Phase.
- Planning Phase.
- Requirement Analysis Phase.
- Design Phase.
- Development Phase.
- Integration and Test Phase.
- Implementation Phase.
- Operation and Maintenance Phase.
- Disposition Phase.

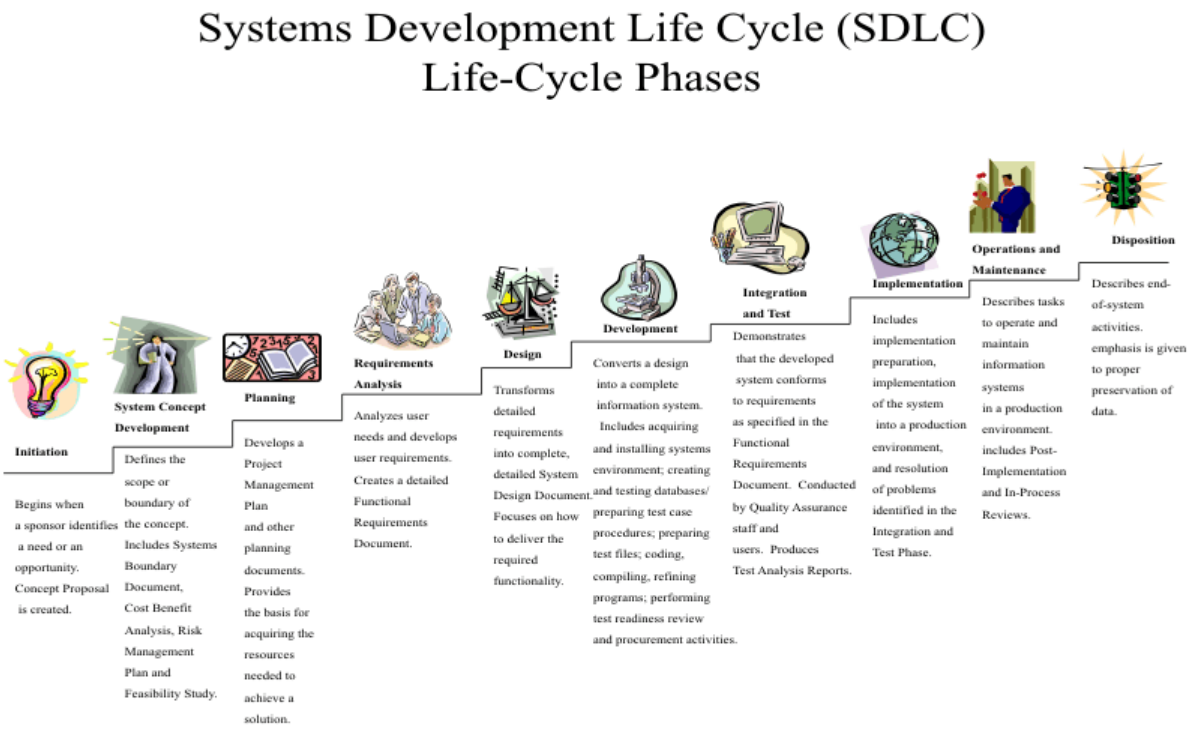


Figure 3.1 The System Development Life Cycle (SDLC)

Initiation Phase

The initiation phase begins when the need to start a project is identified by the sponsor. The requirements of the project are stated and a plan is formed to fulfil those requirements. A clear guideline is formed with objectives and goals. SDLC places great importance on documentation, therefore, each requirement and the plans for their achievement are documented.

Security planning should also be started at this phase. Major security roles should be identified and documented. The Information System Security Officer (ISSO) is appointed and the security information is communicated to team members as well as to the stakeholders. Project requirements are evaluated on the bases of security before making the initial plans for project development. Security is crucial before starting any project as it identifies possible threats and obstacles that may hinder the development process. In addition to security, the availability of resources and project requirements are evaluated and team members are appointed. Making all critical arrangements during initiation helps in cost and time savings and provides a clear plan for project development (Hardware S. R., PHASE 1: INITIATION PHASE).

Goals of the Initiation Phase

The goals of the initiation phase are listed below (Hardware S. R., PHASE 1: INITIATION PHASE).

- Opportunities for business operation's improvement are identified and defined.
- Sponsors for project are identified.
- Project Concept and Project Charter are initiated.
- Project manager is appointed.
- Formation of planning team.
- Approval for the Concept Development Phase.

Approvals and Deliverables

The deliverables of each phase indicate the successful execution and implementation of the process. Additionally, the deliverables of SDLC help state agencies in project planning, execution, and establishment of IT controls. A framework is defined using the deliverables to make sure the objectives and plan are clearly communicated throughout the organization. These deliverables along with their goals are defined below (Hardware S. R., PHASE 1: INITIATION PHASE).

Deliverable	Goals	Developed By	Approved By
Concept Proposal – describes the need or opportunity to improve existing agency business functions using automation and technology. This document identifies unmet strategic goals or mission performance improvements.	• Investigate and document information about the project request • Highlight unmet strategic goals • Identify needed agency mission performance improvements • Obtain commitment to the project from key stakeholders, principally external stakeholders, • Avoid identifying specific vendors or products as solutions	Project Sponsor	Executive Sponsor Agency Chief Information Officer (CIO)
Project Charter – identifies the Project Manager, Project Sponsor, and Executive Sponsor and authorizes the Project Manager to execute the project.	• Develop a broad statement of the purpose of the project • Delineate clear project objectives • Identify the Executive Sponsor, Project Sponsor, and Project Manager	Project Manager	Executive Sponsor Project Sponsor Agency CIO
Project Organization Chart (Draft) – a graphical depiction of the project's hierarchical positions and relationships.	• Identify key project personnel • Define relationships between the team members • Identify required roles	Project Manager	Executive Sponsor Project Sponsor Agency CIO

Figure 3.2 Initiation Phase Deliverables

All possible deliverables, besides the ones already identified, should be included along with the deliverable updates. The deliverables of the initiation phase are revised and modified throughout the development process. Copies of the documentation are prepared in the initiation phase and are distributed among the team members and stakeholders to get approval and feedback.

Participants of Initiation Phase

The people below are part of the initiation phase (Hardware S. R., PHASE 1: INITIATION PHASE).

- Project Stakeholders.
- Project Sponsors.
- Executive Sponsors.

- Agency CIO.
- Business Owner.
- Project Manager.

System Concept Development Phase

The System Concept Development Phase begins after the initiation phase has been completed successfully. The main purpose of the concept development phase is to identify alternative solutions and to clearly define goals, project scope, deliverables, resources and development plans. Any risks associated with the project are identified and evaluated in the concept development phase (Hardware S. R., PHASE 2: CONCEPT DEVELOPMENT PHASE).

Goals of System Development Phase

The goals of the system development phase are listed below (Hardware S. R., PHASE 2: CONCEPT DEVELOPMENT PHASE).

- Business need analysis.
- Project scope definition.
- Alternative evaluation.
- Project risk assessment.
- Roles and responsibilities are defined.
- Initial work breakdown structure.
- Information technology request.
- Project viability determination.
- Approval to proceed to the next phase.

Approvals and Deliverables

The project deliverables provide a clear picture of the project development process. Every deliverable with SDLC is used by state agencies to evaluate and monitor the project

plan, execution and implementation strategy and to provide guidance in case of errors and faults.

Deliverable	Goals	Developed By	Approved By
Project Scope Statement (PSS) – documents the scope of a project and its business case with the high-level requirements, benefits, business assumptions, alternatives analysis, and program costs and schedules.	• Records early research and decisions relevant to the project • Guides project execution to achievement • Describes the boundaries, specifically in and out of scope work, and scope control methods • Establishes a baseline of key characteristics of the project's high level requirements, estimated project costs, and schedule for agencies and sponsors • Specifies acceptance methods and measurable acceptance criteria for all project deliverables • Documents alternatives analysis, including the cost-benefit analysis of alternatives	Project Manager Project Sponsor	Agency Chief Financial Officer (CFO) Agency CIO Project Sponsor Project Manager Executive Sponsor

Information Technology Project Request – serves as the formal budget request for the project and has information provided by the PSS. The ITPR is a mandatory deliverable for this project phase for all MITDPs.	• Ensures alignment of MITDP with the State IT Master Plan (ITMP) and the requesting agency's ITMP • Provides documentation of an IT project investment • Captures status, risk, schedule, funding and cost detail for agency IT project requests • Captures procurement information for agency IT projects • Provides a consistent and repeatable process in support of the State's IT Project Oversight methodology • Ensures uniformity of IT project request submissions	Agency CFO Agency CIO Project Manager	Agency CFO Agency CIO Project Sponsor Executive Sponsor
Project Organization Chart (Update) – a graphical depiction of the project hierarchical positions and relationships.	• Identifies key personnel on the project • Defines the relationships between the team members • Identifies the required roles	Project Manager	Agency CIO Project Sponsor Executive Sponsor
Responsibility Assignment Matrix (RAM) – defines in detail the roles, authority, responsibility, skills, and capacity	• Communicates detailed roles, authority, responsibility, skills, and capacity requirements	Project Manager	Agency CIO Project Sponsor
Project Staffing Estimates – details a preliminary estimate of resources required to complete the project and serves as an input for the project staffing management plan in the next phase.	• Determines preliminary estimates of human resources required to successfully complete project	Project Manager	Agency CIO Project Sponsor
Work Breakdown – provides a preliminary Work Breakdown Structure (WBS) and a WBS Dictionary to define all project activities from planning to implementation	• Determines preliminary definition of all high level tasks to successfully complete	Project Manager	Agency CIO Project Sponsor

Figure 3.3 System Concept Development Phase Deliverables

All deliverables must pass quality tests and must be signed off by their respective authority before moving on to the next stage (Hardware S. R., PHASE 2: CONCEPT DEVELOPMENT PHASE).

Participants of the System Concept Development Phase

The authorities below are involved in the System Concept Development Phase (Custom, PHASE 3: PLANNING PHASE).

- Agency CIT.
- Agency CIO.
- Project Sponsor.
- Executive Sponsor.
- Project Manager.
- Project Stakeholder.
- Planning Team.

Planning Phase

The planning phase primarily focuses on formation of the project development plan. In order to execute the project, a plan is essential. The planning phase identifies all the crucial elements that need to be a part of the planning process. Planning out a project makes the task of project management easier. The planning phase includes identification, definitions, combination, and coordination of all activities into a plan for successful deployment (Custom, PHASE 3: PLANNING PHASE).

Goals of the Planning Phase

The goals below are part of the planning phase (Custom, PHASE 3: PLANNING PHASE).

- Formation of procurement management strategy.
- Identification and evaluation of project risk, project cost, and schedule.

- Definition of activities and sub-activities along with the subsidiary plan.
- Development of procedures to execute, control, monitor and shut down the project.
- Future planning of the project.
- Development of Project Management Plan (PMP).
- Approval to proceed to the next phase.

Approvals and Deliverables

The deliverables of each phase must be revised carefully before proceeding to the next phase. All documents should be modified and updated to reflect any changes (Custom, PHASE 3: PLANNING PHASE).

Deliverable	Goals	Developed By	Approved By
Project Management Plan – • Scope Management Plan • Schedule Management Plan • Cost Management Plan • Quality Management Plan • Staffing Management Plan • Communication Management Plan • Risk Management Plan • Procurement Management Plan • Change Management Plan	• Define how the project is executed, monitored, controlled, and closed • Document all actions necessary to execute, monitor, control, and close the project	Project Manager	Agency CIO Project Sponsor

Figure 3.4 Planning Phase Deliverables

Participants of Planning Phase

The people below are included in the Planning Phase (Custom, PHASE 3: PLANNING PHASE).

- Project Manager.
- Project Sponsor.

- Executive Sponsor.
- Agency CIO.
- Procurement Officer.
- Project Stakeholders.
- Agency CIT.

Requirements Analysis Phase

This phase begins when the previous two phases have been completed successfully and the deliverables have been approved. User requirements documentation from the Planning and Concept Development phase are used here to further analyze and develop the details and user requirements are analyzed to determine their rationality and achievability. Stakeholders are included in the discussion to reach a mutually agreed consensus. The user explains his/her requirements explicitly and the details are documented and signed by the client. The requirements are used to develop the system design (Hardware M. R., PHASE 4: REQUIREMENTS ANALYSIS PHASE).

Goals of the Requirements Analysis Phase

The goals below are achieved by successful completion of requirements analysis phase (Hardware M. R., PHASE 4: REQUIREMENTS ANALYSIS PHASE).

- Approved and clearly defined user requirements.
- System requirements document created along with requirements traceability matrix.
- Planned test activities are created.
- Approval to proceed to the next phase.

Approvals and Deliverables

The deliverables of the Requirements Analysis Phase are detailed below (Hardware M. R., PHASE 4: REQUIREMENTS ANALYSIS PHASE).

Deliverable	Goals	Developed By	Approved By
Concept of Operations (ConOps) - describes the characteristics of the proposed system from the users' perspectives.	• Describe operational scenarios of system functions • Identify modes of operation	Planning Team	Agency CIO Project Sponsor Business Owner Project Manager
System Requirements Document (SRD) - a formal statement of a system's requirements, including, but not limited to: functional requirements, data requirements, system interface requirements, non-functional or operational requirements, and physical requirements.	• Document detailed, measurable, consistent, and comprehensive system requirements • Eliminate ambiguity of expectations regarding system	Planning Team	Agency CIO Project Sponsor Business Owner Project Manager
Requirements Traceability Matrix (RTM) - a table that links requirements to their origins and traces them throughout the project life cycle. Developing the RTM helps to ensure that each requirement adds business value and that approved requirements are delivered.	• Establish a baseline for requirements change control, design, and testing	Planning Team	Agency CIO Business Owner Project Manager
Test Master Plan (TMP) - documents the scope, content, methodology, sequence, management of, and responsibilities for test activities.	• Document and communicate tasks and activities needed to ensure that the system is adequately tested and can be successfully implemented	Project Manager Agency CIO Business Owner	Agency CIO Business Owner Project Manager

Figure 3.5 Requirements Analysis Phase Deliverables

Participants of the Requirements Analysis Phase

The people below are directly involved in the requirements analysis phase (Hardware M. R., PHASE 4: REQUIREMENTS ANALYSIS PHASE).

- Project Manager.
- Project Sponsor.
- Executive Sponsor.
- Agency CIO.

- Procurement Officer.
- Project Stakeholders.
- Agency CIT.

Design Phase

All requirements have been identified, defined and evaluated before the beginning of this phase. These requirements are now designed in such a way as to satisfy the client. User requirements that are identified in the Requirements Analysis Phase are converted here into a System Design Document that defines the system accurately (Custom, PHASE 5: DESIGN PHASE).

Goals of the Design Phase

The goals of the design phase are listed below (Custom, PHASE 5: DESIGN PHASE).

- Converting user requirements into a design for product development.
- Assessment and mitigation of risks.
- Approval to proceed to the next phase.

Approvals and Deliverables

Deliverables must be signed off and approved by their respective authority. All deliverables are documented and updated before signing if changes are made. The System Design Document, System Security Consensus Document, Security Plan and Data Retention Plan are examples of the deliverables of the Design Phase (Custom, PHASE 5: DESIGN PHASE).

Deliverable	Goals	Developed By	Approved By
System Design Document – specifies the construction details of the system, each system component's interaction with other components and external systems, and the interface that allows end users to operate the system and its functions.	• Document the results of the system design process • Describe how the system with satisfy requirements	Development Team	Project Sponsor Agency CIO Project Manager
System Security Consensus Document (SSCD) – a single document containing all information relevant to completing the system's C&A.	• Define the system's security architecture, security policies, risk assessments, and security tests • Consolidate all information for the C&A	Project Manager	Agency CIO
Security Plan – documents the scope, approach, and resources required to assure system security.	• Describe planned activities to control access and protect the system and its information	Development Team	Project Manager Agency CIO
Data Retention Plan – describes the project policies for data and records management.	• Record retention and disposition responsibilities • Document retention and disposition requirements • Record management process • Document retention and disposition schedules	Development Team	Project Manager Agency CIO

Deliverable	Goals	Developed By	Approved By
Disaster Recovery Plan – IT-focused plan designed to restore operability of targeted systems, applications, or a computer facility due to a natural or man-made extended interruption of an agency's business services.	• Identify plans to restore operability in the event of extended interruption of services • Define and document concept of operations • Document notification procedures • Record damage assessment procedures, recovery activities, and reconstitution procedures	Development Team	Project Manager Agency CIO
Unit and Integration Test Plans (Begin) – detailed scripts used in the Development and Test Phases for evaluating the completeness and correctness of the smallest parts of the system and the components created from those parts. The test scripts are more specific than the Test Master Plan, which is high-level and more focused on processes.	• Identify detailed scripts for testing system components	Development Team	Agency CIO Project Manager
Conversion Plan (Begin) – describes the strategies and approaches for converting/migrating data from an existing system to another hardware or software environment.	• Document all planned activities to ensure a smooth data conversion from a legacy system to a new environment	Development Team	Agency CIO
Implementation Plan – describes how the information system will be deployed as an operational system.	• Define all planned activities to ensure successful implementation to production operations	Development Team	Project Sponsor Agency CIO

Deliverable	Goals	Developed By	Approved By
Operations or System Administration Manual (Begin) – The Operations Manual focuses on mainframe systems; the Systems Administration Manual is oriented for distributed (client/server) applications. Both documents provide details on system operations.	• Provide detailed instruction for system operations	Development Team	Agency CIO
Maintenance Manual (Begin) – details effective system maintenance. Appendices might document maintenance procedures, standards, or other essential information on areas such as backup, networking and connectivity, access and authentication, cabling, and critical services.	• Provide maintenance personnel with the information necessary to effectively maintain the system	Project Manager	Agency CIO
Training Plan – outlines training needs for end users on the new or enhanced information system.	• Ensure that the schedule accounts for all necessary training needs to successfully implement, operate, and maintain the system	Development Team	Project Sponsor
User Manual (Begin) – describes to end users how to make full use of the information system, including system functions and capabilities, contingencies and alternate modes of operation, and step-by-step procedures for system access and use.	• Provide users with detailed information to fully utilize the system	Development Team	Agency CIO

Deliverable	Goals	Developed By	Approved By
Requirements Traceability Matrix (Update) – a table that links requirements to their origins and traces them throughout the project life cycle.	• Establish a baseline for requirements change control, design, and testing	Development Team	Agency CIO Business Owner Project Manager

Figure 3.6 Design Phase Deliverables

Participants of the Design Phase

The people below are the design phase participants (Custom, PHASE 5: DESIGN PHASE).

- Project Manager.

- Project Sponsor.
- Executive Sponsor.
- Agency CIO.
- Development Team.
- Project Stakeholders.
- Agency CIT.
- Security Officer.

Development Phase

The development phase can only be initiated if the previous stages have been completed successfully. It requires a complete set of specifications and proper standards and processes. The actual work on the software begins at this stage. The coding is done, functionality is formed and also the database is designed. During the development phase, test functionalities are also prepared (Hardware S. R., PHASE 6: DEVELOPMENT PHASE).

Goals of the Development Phase

The goals below are achieved by successful completion of the development phase (Hardware S. R., PHASE 6: DEVELOPMENT PHASE).

- System building.
- Integration and testing of units.
- Preparation of technical environment for the software.
- Approval to proceed to the next phase.

Approvals and Deliverables

The deliverables of the development phase are used for the purpose of testing. They are reviewed before forwarding them to the security officer. The following are deliverables of the Development Phase (Hardware S. R., PHASE 6: DEVELOPMENT PHASE).

Deliverable	Goals	Developed By	Approved By
System Development Document – establishes the hardware and network development approach including methodologies, tools, and procedures to be employed; also includes development procedures for issue tracking and configuration management and any other information that aids in the implementation of the system.	• Document all preparations related to the development of the system • Describe development methodologies, tools, and procedures	Development Team	Agency CIO
System – integrated hardware, network, and/or firmware components that meet all requirements.	• Provide system that meets the business needs and all requirements	Development Team	Agency CIO
Integration Document – describes the assembly and interaction of the hardware, network, and any other components of the system.	• Document planned approach and activities for the integration of hardware, network, and other system components	Development Team	Agency CIO
Test Analysis Report(s) – presents a description of the unit tests and the results mapped to the system requirements; also identifies system capabilities and deficiencies.	• Record results of tests • Present the capabilities and deficiencies for review • Provide a means for assessing system progression to the next stage of installation or testing	Development Team	Project Sponsor Agency CIO

Deliverable	Goals	Developed By	Approved By
Conversion Plan (Update) – describes the strategies and approaches for migrating data from an existing system to another hardware/network environment. This document is only applicable for projects involving the migration of data.	• Document all planned activities to ensure a smooth data migration from a legacy system to a new environment	Development Team	Project Sponsor Agency CIO
Implementation Plan – describes how the information system will be deployed as an operational system.	• Define all planned activities to ensure successful implementation to production operations	Development Team	Project Sponsor Agency CIO
Operations Manual or Systems Administration Manual – The Operations Manual focuses on mainframe systems; the Systems Administration Manual is oriented toward distributed (client/server) systems.	• Provide detailed instruction for system operations	Development Team	Agency CIO
Release Notes – provides summary information regarding the current release of the system being built; typically includes major new features and changes and identifies known problems and workarounds.	• Document critical information regarding the system release	Development Team	Agency CIO
Maintenance Manual – details effective system maintenance. Appendices might document maintenance procedures, standards, or other essential information.	• Provide maintenance personnel with the information necessary to maintain the system effectively	Development Team	Agency CIO
Training Plan – outlines technical and user training needs on the new or enhanced information system.	• Ensure the schedule accounts for all necessary training needs to implement, operate, and maintain the system successfully	Development Team	Project Sponsor Agency CIO

Figure 3.7 Development Phase Deliverables

Participants of the Development Phase

The members below are included in the development phase (Hardware S. R., PHASE 6: DEVELOPMENT PHASE).

- Project Sponsor.
- Executive Sponsor.

- Project Manager.
- Development Team.
- Project Stakeholders.
- Agency CIO.

Integration and Test Phase

The test phase is designed to test the functionality of the software according to the user requirements. It is not possible to test all possible scenarios of software functionality because there are infinite ways a function can work. Therefore, testing is designed in such a way that it should identify faults and defects, if they exist. Testing is crucial because bugs that are found earlier are easier and less costly to fix. Every functional and non-functional aspect of the software is thoroughly tested to look for any possible faults. Any faults found are rectified before deploying the software to the client (COTS).

Goals of the Integration and Test Phase

The goals of the integration and testing phase are as listed below.

- Testing of security, system and user acceptance with respect to initial user requirements.
- Testing to make sure that the software satisfies all technical, business and stakeholder expectations.
- Testing of operation functions to make sure they are in accordance with the User Manual and Operations Manual.
- Approval to proceed to the next phase.

Approvals and Deliverables

The SDLC testing phase provides a platform to test all the deliverables and documents along with the software to avoid major disasters. Testing also helps in quality management. The deliverables of the testing phase are used in the next phases and the deliverables of the previous phases are tested and updated (COTS).

Deliverables	Goals	Developed By	Approved By
Test Analysis Approval Determination - summarizes the system's perceived readiness and is attached to the Test Analysis Report as a final result of the test reviews.	• Document the perceived production-readiness of the system • Serve as an input to the project Readiness Document described below	Development Team	Project Sponsor Agency CIO
Test Problem Reports - document problems encountered during testing; are also attached to the Test Analysis Report.	• Document detailed results of testing	Development Team	Project Sponsor Agency CIO

Deliverables	Goals	Developed By	Approved By
Information Technology Systems Certification & Accreditation - includes completion of a Security Risk Assessment, Sensitive System Security Plan, Security Operating Procedures, Security Test and Evaluation, and Certification Statements. For SaaS efforts, vendors may provide alternative documentation and certification to provide assurance of the same level of security.	• Assess technical and non-technical safeguards to determine the extent to which the system meets security requirements • Obtain formal declaration by a Designated Approval Authority (DAA) that an information system is approved to operate in a particular security mode using a prescribed set of safeguards at an acceptable level of risk	Development Team	Project Sponsor Agency CIO
Defect Log – tracks and summarizes in a tabular format defects or bugs found during testing. Defects may be documented via multiple commercially available bug tracking tools or manually in a spreadsheet.	• Allow team members to track reported bugs, or defects • Clearly communicate summary of defects found • Record facts regarding known bugs, such as times reported, individuals who reported them, defect statuses, and team members responsible for addressing the bugs	Development Team	N/A – The Defect Log does not require approval.

Deliverables	Goals	Developed By	Approved By
Readiness Document – consolidates summary information regarding the current status of the system and the project and provides decision makers with the information necessary to make a "Go-No Go" decision. It should include a checklist for all work products, User Acceptance Test results, other quality control checks such a peer review, and results of the system walkthroughs.	• Provide information necessary to make the "Go-No Go" decision • Consolidate status information regarding the effective completion of the project and achievement of project objectives and SDLC requirements • Affirm achievement of all deliverable acceptance criteria	Development Team	Agency CIO

Figure 3.8 Integration and Test Phase Deliverables

Participants of the Integration and Test Phase

The team members below play an active role in the integration and test phase (COTS).

- Project Stakeholder.
- Project Sponsor.
- Executive Sponsor.
- Project Manager.
- Agency CIO.
- Development Team.

Implementation Phase

The main purpose of the implementation phase is the successful deployment of the software to the user. Support services like operational staff training and maintenance facilities are also provided. Software that has multiple releases, of course, requires multiple implementation testing phases (Hardware M. R., PHASE 8: IMPLEMENTATION PHASE).

Goals of the Implementation Phase

The goals of the implementation phase are listed below (Hardware M. R., PHASE 8: IMPLEMENTATION PHASE).

- Software Deployment.
- System Training.

Approvals and Deliverables

Any changes made are revised and updated before deploying the software to the client. The implementation phase includes the deliverables below (Hardware M. R., PHASE 8: IMPLEMENTATION PHASE).

Deliverable	Goals	Developed By	Approved By
Complete System – includes all code – modules, components, and libraries – kept in the production version of the data repository.	• Deliver system that meets the business need and all requirements • Deploy system to production environment	Development Team	Agency CIO
System Documentation – includes all technical documentation delivered during the project (e.g. the SDD and System Administration Manual).	• Provide all documentation necessary to effectively operate and maintain the system	Development Team	Agency CIO
Implementation Notice – formally requests approval for system changes made during the Implementation Phase.	• Formally request approval for system implementation	Development Team	Agency CIO Project Sponsor
Readiness Document – consolidates summary information regarding the current status of the system and the project and provides decision makers with the information necessary to make a "Go/No Go" decision. It should include a checklist listing all work products, acceptance test results, other indicators of success measures and deliverable acceptance.	• Provide information necessary to make the go/no-go decision • Consolidate status information regarding the effective completion of the project and achievement of project objectives and SDLC requirements • Affirm achievement of all deliverable acceptance criteria	Development Team	Agency CIO

Deliverable	Goals	Developed By	Approved By
Version Description Document – primary configuration control document used to track and control versions of a system released to the operational environment. It also summarizes features and contents for the build and identifies and describes the version delivered.	• Allow for tracking and control of system releases to the operational environment • Document features and content in system builds • Identify the version of the system being delivered	Development Team	Agency CIO
Post-Implementation Review Report – summarizes the assessment of Implementation activities at the end of the Implementation Phase.	• Summarize assessment of implementation activities • Evaluate the effectiveness of the system development after the system has been in production • Determine if the system does what it was designed to do	Project Manager	Agency CIO Project Sponsor Project Manager Development Team
Standard Operating Procedures (SOP) (Optional) – defines in detail how the Systems Team will perform the business processes related to the operations and maintenance of the system.	• Provide detailed instructions for future business processes • Ensure consistent execution of business processes • Drive performance improvement and improve organizational results	Development Team	Agency CIO

Figure 3.9 Implementation Phase Deliverables

Participants of the Implementation Phase

The people below are involved in the implementation phase (Hardware M. R., PHASE 8: IMPLEMENTATION PHASE).

- Project Stakeholder.
- Project Sponsor.
- Executive Sponsor.
- Project Manager.
- Agency CIO.

- Development Team.

Operations and Maintenance Phase

This phase is designed to maintain the software quality and performance after it has been deployed to the client and it continues unless a new version is released or the software product is terminated. Maintenance is essential for the smooth running of the software. Any defects found at the user's end are rectified and a new version is release. Guidance is provided to the client to make him/her understand the functionality of the software easily (Hardware S. R., PHASE 9: OPERATIONS AND MAINTENANCE PHASE).

Goals of the Operations and Maintenance Phase

Below are the goals of the operation and maintenance phase (Hardware S. R., PHASE 9: OPERATIONS AND MAINTENANCE PHASE).

- Change management for the system to support the end user.
- Continuous monitoring of software performance.
- Maintenance of security activities like backups, audits and planning.
- Assistance of the end user through support and training.

Approvals and Deliverables

The operation and maintenance deliverables are listed below (Hardware S. R., PHASE 9: OPERATIONS AND MAINTENANCE PHASE).

Deliverable	Goals	Developed By	Approved By
Standard Operating Procedures (Updated) – defines in detail how the Systems Team will perform the business processes related to the operations and maintenance of the system.	• Provide detailed instructions for future business processes • Ensure consistent execution of business processes • Drive performance improvement and improve organizational results	Systems Team	Agency CIO
Performance Reports – tracks routine metrics as system performance indicators.	• Report on agreed upon system performance measurements • Include key performance indicators	System Manager	No approval required
Implementation Notice – formally requests approval for system changes made during the Implementation Phase.	• Formally request approval for system implementation	Project Manager	Agency CIO
Program Trouble Reports – provide details regarding an incident related to any aspect of an IT service.	• Document and track system incidents • Communicate need to address a disruption in service and/or a reduction of quality of service	Systems Team	No approval required
In-Process Review Report – formally reports the health of the system. It includes summary of performance reports but is more formalized and usually developed quarterly.	• Provide Agency CIO with routine insight into system performance • Include results of user satisfaction reviews	System Manager Agency CIO	Agency CIO
User Satisfaction Review – determines the current user satisfaction with the performance capabilities of the system.	• Quantify user satisfaction levels	System Manager	Agency CIO
Disposition Plan – identifies how the termination of the system/data will be conducted, and when, as well as the system termination date, system components to be preserved, disposition of remaining equipment, and archiving of life cycle products.	• Address all facets of archiving, transferring, and disposing of the system and data	System Manager	Agency CIO Business Owner

Figure 3.10 Operation and Maintenance Phase Deliverables

Participants of the Operation and Maintenance Phase

The people below are involved in the operation and maintenance phase (Hardware S. R., PHASE 9: OPERATIONS AND MAINTENANCE PHASE).

- Project Stakeholder.
- Project Sponsor.
- Executive Sponsor.
- Project Manager.
- Agency CIO.
- Development Team.
- Process Improvement Review Board.
- Security Officer.
- Procurement Officer.

Disposition Phase

The disposition phase is the end of the SDLC development process. All activities are terminated. The documentation of the development process is kept in a safe place for future references and the termination of the system is concluded according to organizational laws. The system performance is then evaluated to see whether the software was a success or not (Hardware S. R., PHASE 10: DISPOSITION PHASE).

Goals of the Disposition Phase

- Notification to client.
- Notification to the development team and stakeholders.
- Shutting down of system.
- Disbanding of team and resources are freed up.

Approvals and Deliverables

At the end of the SDLC process, very few deliverables are produced. A final document is created to record the termination process (Hardware S. R., PHASE 10: DISPOSITION PHASE).

Deliverable	Description	Developed By	Approved By
Disposition Plan (Update) – identifies how the termination of the system/data will be conducted, and when, as well as the system termination date, system components to be preserved, disposition of remaining equipment, and archiving of life cycle products.	• Address all facets of archiving, transferring, and disposing of the system and data	System Manager	Agency CIO

Figure 3.11 Disposition Phase Deliverables

Participants of the Disposition Phase

- Business Owner.
- Agency CIO.
- System Manager.

SDLC Roles and Responsibilities

To complete the project successfully and within the triple constraints, there has to be proper coordination and collaboration between the Agency Project Sponsor, the State Chief of Information Technology (CIT), and the Agency Chief Information Officer (CIO). The responsibility of the sponsor is to identify business needs and priorities. The Agency CIO determines the best possible way to implement the technology and the State CIT approves the funding and provides guidance (Systems Development Life Cycle Volume 1 Introduction to the SDLC, 2006).

The role and responsibilities according to each phase are described below (Systems Development Life Cycle Volume 1 Introduction to the SDLC, 2006).

Agency Chief Information Officer (CIO)

The agency CIO is the main advisor of the process who makes sure that IT resources and technology is utilized correctly. The agency CIO works alongside the project manager to administer the project development process according to the defined IT standards.

The Agency CIO ensures that quality standards and customer requirements are fulfilled effectively. He/she is also responsible to provide guidance, opinion, and help as needed (Systems Development Life Cycle Volume 1 Introduction to the SDLC, 2006).

Responsibilities According to the SDLC Phases (MaryLand)

Initiation

- Appointment of the project sponsor to assist the business owner and the executive sponsor.
- Approval of the final proposal for the project with the project sponsor.
- Authorization of the project charter with the project sponsor.
- Provide input to the project sponsor regarding the project charter.
- Revision of the project organization charts and feedback accordingly.
- Presence at initial phase status review.
- Sign off of deliverables.

System Concept Development

- Provides support to the project sponsor and the project manager in the development of the project scope statement.
- Contact type recommendations are reviewed with the project sponsor, the planning team, and the executive sponsor.
- Defines the governing framework of the development life cycle and establishes steering committee.
- Approval of the project scope statement.

- Approval and revision of the responsibility assignment matrix, project organizational chart, and the estimated project labor.
- Presence in concept development phase status review.
- Sign off of phase deliverables.

Planning

- Presence at the planning phase status review.
- With the assistance of the project sponsor, signs off the project management plan that includes all plans required for project execution.

Requirement Analysis

- Approval and revision of the functional requirements documents with the help of the business owner, the project sponsor, and the project manager.
- Approval and revision of the requirements traceability matrix.
- Development of the project proposal for evaluation criteria.
- Getting approval of the evaluation criteria from the agency evaluation committee.
- Approval and revision of the test master plan.
- Organizes the requirements analysis phase status review with the business owner and the project sponsor.

Design

- Approval and revision of the project design documents with the project manager and the project sponsor.
- Approval and revision of the system security document.
- Approval and revision of the security plan and the data retention plan.
- Integration test plan revision and feedback.
- Approval and revision of the implementation plan.

- Revision and feedback of the draft user manual.
- Approval and revision of the updated requirements traceability matrix.
- Design phase status review meeting with the stakeholders.

Development

- Approval and revision of the hardware/network documentation.
- Approval and revision of the system application software.
- Approval and revision of the integration testing document and test and analysis report.
- Approval and revision of updates to multiple documents.
- Approval and revision of the release notes.
- Development phase status review meeting with the stakeholders.

Integration and Test

- Approval and revision of the test analysis approval determination.
- Approval and revision of changes made to any component of the software.
- Approval and revision of test problem reports.
- Makes “Go-No-Go” decisions related to the software.
- Test phase status review conduction with the stakeholders.

Implementation

- Approval of project deliverables including the complete system and documentation.
- Presence in the post-implementation review.
- Approval and revision of the post-implementation review and standard operating procedures.

- Implementation phase status review with the stakeholders.

Operations and Maintenance

- Approval and revision of the updated standard operating procedures.
- Quarterly in-process review with project manager's help.
- Implementation notice approval and revision.
- Approval and revision of user satisfaction review, disposition plan, and change requests.
- Operation and maintenance phase status review with the stakeholders.

Disposition

- Approval and revision of the updates to the disposition plan.
- Disposition phase status review with the stakeholders.
- Signs off post-terminated review reports.

Agency Project Sponsor

The project sponsor is a person responsible to provide direction to the project. He/she acts as spokesperson for the overall project. It is the responsibility of the project sponsor to ensure that the objectives of the project are aligned with business accomplishments and needs. The project documents are scrutinized by the project sponsor to ensure that the project is designed to meet the functional and non-functional requirements.

The project sponsor is responsible for the timely provision of business and financial resources. He/she has an active role in the development process and is responsible to provide assistance regarding risk management and other problematic issues (Systems Development Life Cycle Volume 1 Introduction to the SDLC, 2006).

Responsibilities According to the SDLC Phases (MaryLand)

Initiation

- Development of the concept proposal with the help of the business owner.

- Appointment of the project manager.
- Appointment of the planning team with the help of the project manager.
- Approval and revision of the draft project organizational chart.
- Provide input to the project manager for the project charter.
- Attends initiation phase status review.
- Signs off deliverables.
- Accountable for execution of the project.

System Concept Development

- Provide input for the project scope statement.
- Provide input and approval of the ITPR (information technology project requests)
- Acquisition strategy revision.
- Defines the framework for governing activities, forming of the steering committee, the project organization chart, and the responsibility matrix.
- Head of steering committee.
- Creates the project staff estimation with the project CIO.
- Revision of contract type.
- Attends the concept development phase review.
- Signs off deliverables.

Planning

- Risk identification and mitigation.
- Approval and revision of all accuracy and completeness plans.
- Signs off and revises reasons for early termination (If applicable)
- Attends the planning phase status review.

Requirement Analysis

- Functional requirements documents revision and approval.
- Approval and revision of procedural documents related to the project.
- Attends requirement analysis status review.

Design

- Approval and revision of the System Design document.
- Approval and revision of the training plan.
- Approval and revision of the implementation plan.
- Attends design phase status review.

Development

- Revision of the test problem reports.
- Revision of the defects log and readiness documents.
- Attends development phase status review.

Integration and Test

- Revision of the test problem reports.
- Revision of the test analysis approval determination.
- Assists in systems “go-no-go” decisions.
- Attends test phase status review.

Implementation

- Approval of changes made to the project.
- Approval of the implementation notice.
- Attends post-implementation review.

- Attends implementation phase review.

Operations and Maintenance

- None.

Disposition

- None.

Chief of Information Technology (CIT)

The Chief of Information Technology (CIT) is responsible for providing oversight over the state agencies. This includes, providing guidance related to implementation of SDLC, peer review, feedback and review regarding ITPR, execution of the validation and verification review, approval of project funding, and project management oversight (Systems Development Life Cycle Volume 1 Introduction to the SDLC, 2006).

Responsibilities According to the SDLC Phases

Initiation

- Revision of project documents including the project charter, the concept statement, the project organizational chart, and the responsibility matrix.

System Concept Development

- Revision and approval of the information technology project request.
- Revision of the project documents including the scope statement, the project organizational chart, the responsibility matrix, and the project staffing estimates.
- Revision of the project reporting monthly and quarterly.

Planning

- Revision of the project management plan and the subsidiary plan.
- Feedback on revisions.
- Revision of the project reporting monthly and quarterly.

Requirements Analysis

- Functional requirements document review.
- Revision of the project reporting monthly and quarterly.

Design

- Project documents revision and feedback.
- Revision of the project reporting monthly and quarterly.

Development

- Project documents revision and feedback.
- Revision of the project reporting monthly and quarterly.

Integration and Test

- Project documents revision and feedback.
- Revision of the project reporting monthly and quarterly.

Implementation

- Project documents revision and feedback.
- Revision of the project reporting monthly and quarterly.

Operations and Maintenance

- Project documents revision and feedback.
- Revision of the project reporting monthly and quarterly.

Disposition

- None.

Advantages of SDCL

Listed below are advantages of SDLC.

- Maximum management control as each deliverable is formally reviewed at the end of each phase.
- Documentation of each activity provides concrete evidence.
- Tracking is easy as everything is clearly documented.
- The deliverables are produced and reviewed. If there are any errors, then they are rectified. Resolving errors quickly is easier than reviewing the complete project at the end. Problems found at the end of the project are more expensive and time consuming to resolve.
- Systematic approach to software development.
- Training and maintenance services provided.

(Advantages and Disadvantages of SDLC, n.d.)

Disadvantages of SDLC

Listed below are disadvantages of SDLC.

- Lack of communication with the end user.
- End user only sees the final product.
- At times it is hard for the user to comprehend the functionality of the software.
- Documentation is expensive and time consuming. It may also become obsolete rapidly as technology is always changing.
- Users have no say in the development process nor do they get to review the intermediate products.
- Does not encourages changes. It is rather a riding development process.

(Advantages and Disadvantages of SDLC, n.d.)

Chapter 4

Capability Maturity Model

Software development has never been an easy task. A lot of research has been conducted to find ways to make software that fulfils the requirements as well as produces required results. Throughout the years many solutions have been proposed to solve the problem of software development and one of these many solutions is the Capability Maturity Model also known as CMM. The Capability Maturity Model was originally developed by the Software Engineering Institute (SEI) at Carnegie-Mellon University to improve the process of software development. The term CMM is also used occasionally to describe other improved software development approaches but here our main focus is on the SEI model (Hurst).

The purpose of a process improvement model is to define the set of steps needed to plan, implement, and evaluate business processes adopted by an organization. CMM clearly defines these steps for the ease of software development as well as for the organizational evaluation process. Application of predefined steps not only makes the process run smoothly but also saves time. CMM defines 5 maturity levels for the software development process. It strives to regulate the development process that is otherwise very unorganized and chaotic and it is designed in a way to regulate the development process with efficiency and effectiveness.

After the original introduction of CMM a more sophisticated approach was created and named the Capability Maturity Model Integration (CMMI). This was in the year 2002. Both of these models will be discussed further in this chapter.

History of CMM and CMMI

The development of the CMM process started in August 1986 by the Software Engineering Institute (SEITM) at Carnegie Mellon University with the help of the MITRE Corporation. The goal was to develop a process that would help organizations develop software with ease. The CMM effort was initiated after a request from the Federal Government asking for assistance in software development methodologies. In June 1987, SEI released a brief description of CMM and what it intended to do and in September 1987, the initial maturity questionnaire was released. Based on the initial release and on the answers to

the questionnaire CMM was completed (Paulk, A History of the Capability Maturity Model for Software).

The first version of CMM was released in 1991. It was known as CMM Version 1.0. The second version called Version 1.1 was later released in 1993 and a book was published on it in 1995. The Capability Maturity Model has not been used much after it was modified to be Capability Maturity Model Integration (CMMI) but it has inspired the formation of many frameworks and standards such as People CMM, System Security Engineering CMM, and the Systems Engineering CMM (Paulk, A History of the Capability Maturity Model for Software). It has also played an important role in the formation of ISO/IEC 15504 (Process Assessment), especially the part 7- assessment of organizational maturity (ISO 2008).

CMM software development techniques have been a revolutionary invention that left a great impact on the software development community. Billions of dollars were spent on its formation and maintenance throughout the years. Research papers, case studies, and numerous reports have been published on CMM's popularity as well as its functionality.

The need for CMM ultimately arose in the 1960s, when the US military desired to have a perfect harmonic combination of hardware and software. It was crucial because military and aerospace is responsible for the protection of the entire nation and any fault in the system could result in the loss of thousands of lives. Therefore, a reliable system was needed. This resulted in the creation of CMM by SEI (Software Engineering Institute) that still manages CMM and its integrated version CMMI (SidKemp, n.d.). The main objective was to ensure successful execution of US military and NASA projects because precious human lives depend on these kinds of projects. However, the scope of CMM and CMMI is not limited to Military and NASA use only. It can be used in business operations as well to ensure customer satisfaction by meeting requirements efficiently and effectively (SidKemp, n.d.).

Introduction of CMM made the work of US military and NASA easier than it had been and it enabled them to execute projects successfully therefore resulting in customer satisfaction and a relationship of mutual trust with the customer. The maturity levels defined by CMM provide a clear guideline to the delivery team at every executive level on how to plan and execute the project. CMM is not just for complex projects. It can also be used for simple business projects providing a clear description of every level. The objective was to come up with a model that fulfilled the criteria below.

- Based on actual practices.
- Reflected best state of the practices.
- Met the requirements of individuals performing software process assessment, software process improvement, or software capability assessment.
- Provided all necessary documentation.
- Publicly available for use.

CMM Levels and Components

CMM is composed of five maturity levels. Each maturity level provides a foundation for continuous improvement of the development process. The set of goals in each level, when satisfied, results in customer satisfaction, thereby making the development project a success. Completing each level successfully also increases overall capability of an organization (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

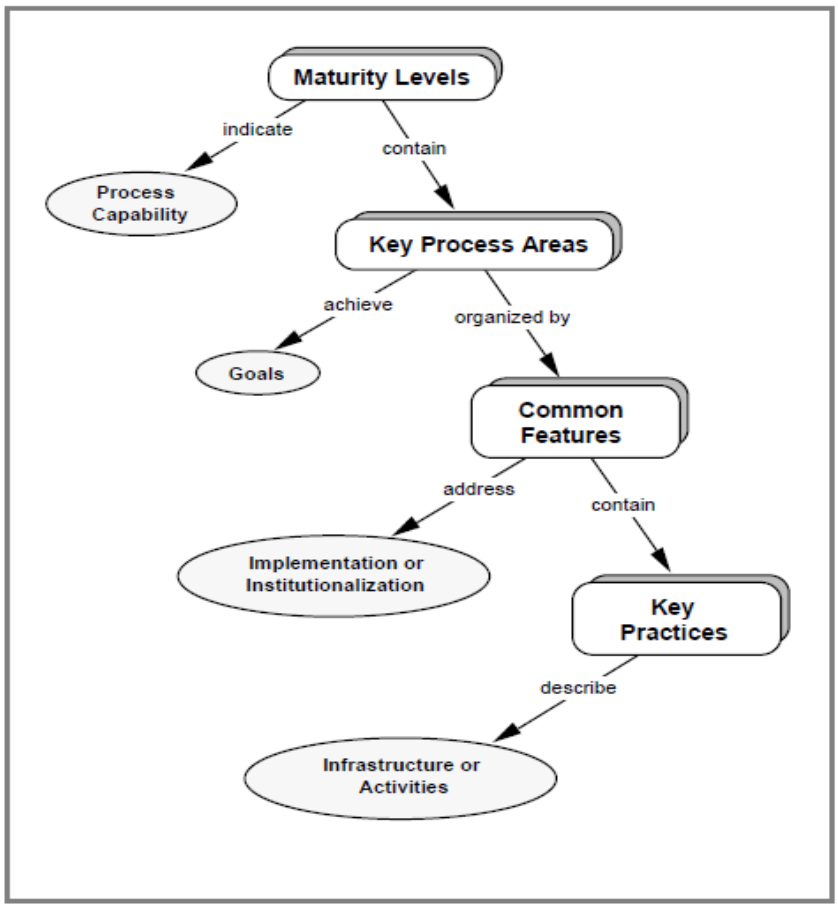


Figure 4.1 CMM Components

Each maturity level is comprised of several key processes with the exception of level 1. The key processes are then further divided into five sections known as common features. The common features define common practices that are employed collectively with key processes to achieve the common goal of the project. This is illustrated in figure 4.1 above (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Maturity Levels: A Maturity Level is a well-defined evolutionary step towards improving the overall project development process. The five well-structured and well-defined maturity levels of CMM provide the delivery team with a clear vision of how to proceed with the development project yielding the best results (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Process Capability: Process Capability describes the expected results that can be achieved if the development process is followed as prescribed. The software development process of any organization provides insight into an organization's work as well as helps in the prediction of the organization's future success or failures (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Key Process Areas: Each maturity level is comprised of key processes. These key processes are a set of related activities combined together to achieve the common goal set by the capability maturity model (i.e. customer satisfaction). The key processes are defined at each maturity level to provide a clear guideline to the delivery team and to aid in the successful completion of the project within the time, resources, and budget constraints (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Goals: The goals provide a common aim for all the project team members to work towards. They summarize the key practices of the key processes that are used to determine the success and failure of the project. The goals define the scope and boundaries and are limited to each key process (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Common Features: The key features are further divided into five common features namely, Ability to Perform, Commitment to Perform, Measurement and Analysis, and Verifying Implementation. These common features indicate whether the project is being developed according to the set standards or not. The Activity Performed common feature indicates the implementation process being employed whereas the rest of the common

features indicate institutionalization factors, which makes the development process a part of the organizational culture (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Key Practices: The key practices are described in order to ensure the successful implementation of key processes. They describe the infrastructure and activities for effective institutionalization of the developmental process (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Five CMM Levels

CMM is based on five maturity levels each used to describe the maturity of the software development process employed by the organization. It aids in the future prediction of an organizations performance based on its current operability. The five maturity levels are described in detail below.

- Initial.
- Repeatable.
- Defined.
- Managed.
- Optimizing.

The levels are illustrated in the figure below (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

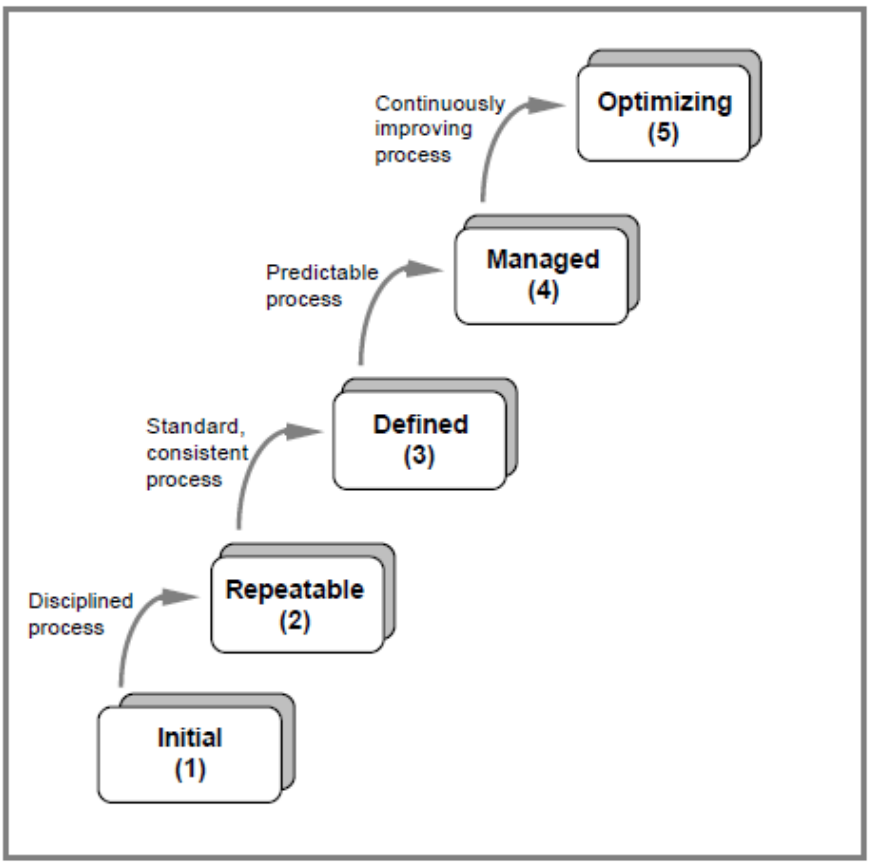


Figure 4.2 CMM Levels

Level 1: Initial

The initial level requires a detailed description of the project that needs to be carried out using CMM. It focuses more on "ad hoc" endeavors as requirements are not static throughout the project delivery life cycle. There is always room for changes because as the project progresses more and more people get involved increasing the complexity level. It supports the changes that take place in an organization as it grows over the years. At this level the involvement of management is highest but due to lack of communication reaching a mutually agreeable decision can be difficult. This causes the end result to be unpredictable thus possibly disappointing clients.

At level one the job of management is to improve and enhance the organizational process and to make room for changes within the organization. Reorganization takes place at this level. It implies that changes made within an organization are communicated and implemented. The team members are selected and rejected according to their abilities and professional skills. Reorganization may cause an increase in the productivity level of

employees but it usually doesn't last long because the increase may be due to stress or unemployment fears of the employees (Capability Maturity Model).

Organizations usually are unable to provide a stable environment for the software development process as they lack effective management personnel and skills to plan and manage the software development project with efficiency. Additionally, organizations aim at the formation of a perfect team with the desired skillsets to develop software according to the client's requirements and specifications. However, the initial process is not stable as there are often changes within the management team that might take place without warning.

The initialization process is also unstable because software requirements tend to change over time. They are not static. Therefore a sound management team is crucial for the success of software development. Also, the right skillsets incorporated in the team will make the process run more smoothly as the team members will understand the requirements and will know what to do and when it should be done. It is important for the organization to opt for a process that is in harmony with the organizational culture in order to avoid conflicts and wasted time (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Level 2: Repeatable

At level two, the delivery team works on the formation of policies, standards, and procedures for the implementation of their development process. Historical data is analyzed to look for similar projects and their success rate. One of the objectives of Level 2 for the management team is to institutionalize effective management processes that enable the organization to repeat past successes. However, practices are not employed as is; changes are made as no two projects are completely identical in nature. The effectiveness of a project can be measured by documentation, practice enforced, team members' skills, and the ability of the organization to incorporate changes and improve gradually (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Realistic guidelines and project scope are established based on the project's requirements and on the past similar project scope. The software project manager then calculates the project cost and makes a rough estimation of the overall expenses that are expected to be incurred throughout the delivery of the project. Meetings are conducted with the stakeholders to reach mutual agreement and gain the consent of each team member. All the requirements are discussed and updated before finalization of the development standards.

After, the standards for software development are defined and the organization ensures they are strictly followed by the delivery team (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Hence level 2 can be described as the standard formation stage where the policies of monitoring and executing the project are established and agreed upon by all team members. It is the responsibility of the project manager to ensure successful execution of the project under the guidelines instituted (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Level 3: Defined

At level 3 the processes for software development are documented including both software engineering as well as management. Documentation is necessary to avoid ambiguities among the team members related to the goal as well as their respective responsibilities. The documentation process is not a rigid one; it can be altered according to the changes incorporated by the client or by the developing team. This is to ensure that the technical staff and software manager perform their responsibilities effectively. Once the standards have been defined and documented, an organization wide training program is executed to ensure every individual working on the project is aware of their responsibilities and goals of the overall project (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

The usual organizational process for software development is altered to suit the needs of the project, but the process is always in accordance with the organizational culture to avoid conflicts. In CMM the altered project is referred to as the project's defined software process. A well-defined process is clear of all ambiguities and contains an integrated and coherent set of defined software engineering and management practices. It can also be categorized as having clearly defined inputs, standards and procedures for work performance, readiness criteria, verification procedures - peer reviews, outputs, and project completion criteria. A well-defined process allows management to perform efficiently because they will clearly understand the process being employed (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Hence, level 3 can be described as where software management and engineering activities are detailed and carried out. They are consistent, stable, and repeatable. Software quality is tracked within the defined cost, schedule, product lines and functionality controls. The process ensures organization wide understanding of the project, activities, responsibilities, and roles to be carried out by each individual (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Level 4: Managed

At this level management initiates measureable quality goals for the software project as well as the software product. It is essential to measure the quality of the product throughout the production process to meet client requirements and hence achieve customer satisfaction. Data is collected and analyzed using an organization wide software project database from the project's defined software process. These standards help measure the success or failure of the development project and make evaluation easier (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

By defining acceptable and unacceptable boundaries, the project maintains control of the development process and team members avoid work rejection. These control limits drive work to be completed efficiently and effectively under continuous monitoring by the project manager. Any variations occurring throughout the project are handled carefully to avoid any irreparable damages to the project. The risks involved are also anticipated and risk mitigation strategies are formed (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Level 4 of the capability maturity model can be described as a predictable measure to control the project's environment and handle risks that might be present throughout the development process. This controlled measure allows organizations to choose the projects that are most likely to succeed under the watchful eyes of the project manager and that are aligned with the organizational culture. This also enables the organization to predict future trends of success and failure using historical data as well as evaluate their current projects. If the project begins to suffer from scope creep precautionary measures are taken to avoid failure and rectify discrepancies. Software products are generally high quality products therefore their continuous monitoring is essential for success (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Level 5: Optimizing

At level 5 the organization focuses on project management improvement. The organization identifies project management strengths and weakness and works towards achieving the project goals without the occurrence of defects. Cost benefit analysis is conducted using the data acquired during the software development project to propose any technological changes. Innovative ideas that exploit the strength of the software development process can be incorporated throughout the organization so that others can benefit from

effective software management and development (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

At level 5 the management team identifies defects and their causes. These defects and errors are evaluated and analyzed to avoid them from occurring again in the future. The lessons learned from this project are also used as a reference for later projects in the future. This keeps the team from making the same mistake over and over again (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

The software development process at level 5 can be described as continuously improving as organizations at this level are continuously trying to find better ways to improve their project's efficiency, capability, and performance. Improvements are made in internal processing as well as external processing and both the technological aspect and management of the process are continuously improved to achieve better results and increase customer satisfaction (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Key Processes of CMM

Key processes are a collection of related activities that when performed collectively result in the achievement of project goals. They're defined after level 1 because the rest of the levels are slightly more mature. The key processes are the main focus of any organization as they are the building blocks of the software development process being employed. Figure 4.3 below illustrates the key processes involved at each CMM maturity level (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

Key Processes at Level 2

At level 2 the key processes focus mainly on the establishment of project management controls. Each of those key processes is described briefly below (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

- **Requirements Management:** focuses on establishing a better understanding between the customer and the software project delivery team. Customer's agreement is required before processing the development project. All basic requirements are clarified and Software Configuration Management is established to incorporate future changes.

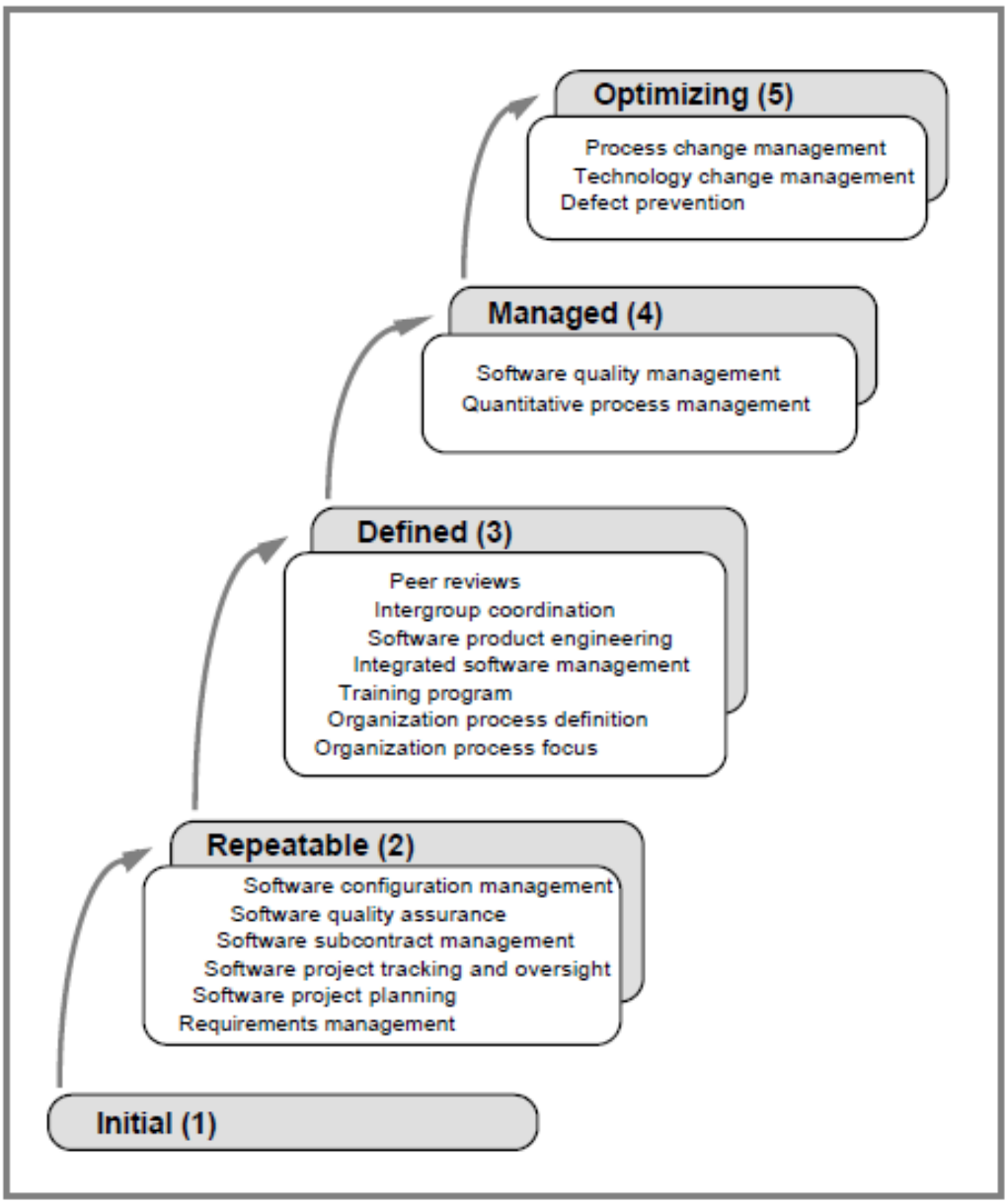


Figure 4.3 CMM Key Processes

- **Software Project Planning:** aims at establishing plans for software management and engineering throughout the development project. Realistic and attainable plans and goals are necessary for effective execution of the project.

- **Software Project Tracking and Oversight:** keeps track of the development activities being performed by the team. Any deviation from the actual plan is documented immediately and appropriate measures are taken to rectify the issue.

- **Software Subcontract Management:** the main purpose of Software Subcontract Management is to employ qualified subcontractors who are able to work on the

project effectively. It combines all the key process and applies them to the subcontractors as appropriate.

- **Software Quality Assurance:** aims to ensure quality standards are followed and the final product meets customer requirements.
- **Software Configuration Management:** the purpose is to maintain software integrity and to incorporate changes that may occur in the future.

Key Processes at Level 3

At level 3 the product and organizational issues are addressed. The key processes at level 3 are briefly discussed below (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

- **Organization Process Focus:** aims at establishing organization responsibilities related to the project and to improve the overall project management process within the organization.
- **Organization Process Definition:** aims to develop an effective software development process that can be employed whenever needed and that contributes to the improvement of the organization as a whole.
- **Training Program:** aims to provide training to employees to maintain their skills so that they can work efficiently and effectively on the project. Training also helps in the reduction of human error.
- **Integrated Software Management:** the aim of Integrated Software Management is to integrate software management and engineering activities in a coherent way that can be related to the organization's development process and organizational culture.
- **Software Product Engineering:** aims to perform all the engineering tasks efficiently and effectively. It basically addresses the technical aspect of the software such as code, design, analysis, requirements, and test.
- **Intergroup Coordination:** the aim of Intergroup Coordination is to develop smooth communication flow between different engineering groups within the

organization. Failure to communicate effectively will result in project failure and customer dissatisfaction.

- **Peer Reviews:** is used to reduce defects before delivering the final product to the client. It is one of the most effective methods employed to reduce defects and it also promotes a better understanding of the software.

Key Process at Level 4

Level 4 aims at establishing a better understanding of the software development process and software products with quantitative measurements. The key processes at level 4 are described briefly below (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

- **Quantitative Process Management:** aims to measure the performance of the software development process quantitatively. The purpose is to look for aspects that cause variation in the development project and develop a resolution strategy.

- **Software Quality Management:** aims to evaluate the quality standard by quantitative means and to ensure all quality goals are being met to satisfy the customer. Any negligence is immediately reported and the resolution effort is initiated. Each software product must pass quality control before it is delivered to the customer.

Key Process at Level 5

At level five the organizational as well as the project's issues are collectively addressed. Continuous improvement measures are carried out to yield better productivity. Key processes at level 5 are discussed briefly below (Paulk, Weber, Garcia, Chrissis, & Bush, 1993).

- **Defect Prevention:** the causes of defects are identified and investigated. It is ensured that these defects do not occur in the future by taking preventative steps.

- **Technology Change Management:** purpose is to incorporate changing technologies in a timely manager into the organization. Any advances in technology, tools, and methods are documented and embedded in the current organizational process to produce superior results.

- **Process Change Management:** The main objective of Process Change Management is to strive to continuously improve the development process within an organization. It also focuses on the improvement of software quality, decreasing the project delivery life cycle, and increasing productivity levels.

Key processes are defined individually at each maturity level of CMM but they are interrelated as they have an effect on each other. Therefore, it is highly important for an organization to focus on key processes at each maturity level. Organizations may focus on achieving the higher maturity levels first but that doesn't mean they should ignore the lower maturity levels. Each key process is important as it contributes to the overall development of an organization.

Future of CMM

Creating a development process that is efficient and effective is not completed in a matter of days. In fact, sometimes it takes years and even decades to establish a process that can cater to the needs of every kind of software development project. Likewise CMM is not a perfect solution for all projects. There are certain areas that CMM does not address. For example, it does not address the issue of how to hire and retain potential employees for a project and it does not focus on particular domains or specific technologies related to software development. These issues are crucial for the success of any project yet they are not addressed by CMM (Paulk, Curtis, Chrissis, & Weber, 1997).

Since 1986 CMM has evolved and made significant advances in the software development process and will continue to do so. CMM has made a significant difference in the fields of software capability assessment, process improvement programs, software process assessment, and in changing the overall business environment (Paulk, Curtis, Chrissis, & Weber, 1997).

SEI is also working with ISO (International Standard Organization) to further enhance the productivity level of CMM. The joint collaboration of these two companies is expected to yield better results in the future and further improve CMM to cater to the needs of varied IT and Business development projects (Paulk, Curtis, Chrissis, & Weber, 1997).

Advantages of Capability Maturity Model

Listed below are advantages of CMM.

- Integrated Quality Management System creates consistency in documentation.
- Ensure best practices by incorporating software engineering into the overall organization.
- Leads to fewer errors and ultimately less maintenance cost.
- Cost effective.
- Enhances productivity and on-time delivery.
- Flexible execution and ensures effortless communication.
- Comprehensively covers all areas of product life cycle.

Disadvantages of Capability Maturity Model

Listed below are disadvantages of CMM.

- Rigid step by step structure.
- Organizational focus at any time is only on the task at hand.
- Requires added knowledge and resources to implement in smaller companies.
- May require additional time and effort to be effective after implementation.
- Success is generally dependent on organization's size and often calls for culture change.

(Atwal, 2008) (Consultant, 2013) (Disadvantages and Advantages of CMMI-DEV) (Moksony, 2014) (What is CMMI? Explain the advantages of implementing CMMI - CMMI)

Chapter 5

TenStep Project Management Process

Introduction

TenStep is yet another systematic approach to project management. As with other project management life cycles, it has a clearly defined step-by-step procedure for project management and delivery. Just like the name suggests, it includes ten important steps for managing projects. TenStep is a measureable and flexible approach to project management that is suitable for all sizes and kinds of projects. Each step of the process is defined in detail with clear objectives and deliverables. Applying the TenStep Project Management Process saves money and time as it provides a clear road map for the delivery of the project.

Every project, whether large or small, needs management to yield the required results. As the project gets bigger and more complex, managing it can be difficult. In order to manage projects effectively and efficiently the TenStep Management process was developed. From project initiation to project deployment, each step has a clear work approach defined. The sequence of steps can be altered to suit project needs and some projects may skip a step or two altogether depending on the requirements.

History of TenStep

The TenStep Project Management Process was developed by TenStep, Inc. in early 2000. Since then the process has gained in popularity because it’s easy to understand and implement. The process is available to users free of cost. Additionally, there are multiple online tutorials and websites providing guidance on how to use and implement the TenStep Project Management Process as well as books, seminars, and short courses.

The TenStep process is used internationally to assist in project management. It is endorsed by PMI (Project Management Institute) and is registered with the Davinci Institute of Technology. TenStep is a registered trademark.

What is TenStep?

The TenStep Project Management Process is a methodology incorporating the ten most important steps for management of projects. These steps define the project scope, the

project plan, risk assessment, risk control, implementation, and execution of the project. Each step is defined clearly with the help of documentation and the TenStep Project Management Process incorporates project management controls and disciplines. It helps organizations with the successful implementation of their goals and strategies through a systematic and defined framework.

The TenStep process defines each step according to the nature of the project. Project requirements are evaluated and analyzed before they are converted into an executable project plan. Additionally, it provides the basic information needed by the project manager for successful project management by defining activities of each step explicitly.

The TenStep Project Management Process is measurable and flexible. Measurable implies that the project delivery progress can be evaluated and measured on the basis of the clear framework defined by TenStep. Flexible, on the other hand, implies that the TenStep process has the ability to suit any kind and size of project. The life cycle facilitates risk evaluation and allows full control by the project manager. Additionally, like all life cycles, it assists in the management of the scope, time, and resources of the project.

The first two steps of the TenStep life cycle are initiation and planning. Steps 3-10 are for management and control. Each of these steps will be discussed later in this chapter.

Objectives of the TenStep Project Management Process

The TenStep process includes the objectives listed below.

- Definition of the work. All the activities related to the project management process are defined including deliverables.
- Schedule and budget for the project are defined.
- Scope and change management.
- Efficient communication management.
- Risk management and mitigation.
- Human resource management.
- Project quality management.

- Project delivery management.

The TenStep Project Management Process defines the above mentioned objectives to execute the project management plan. The process is categorized as ‘successful’ only when all of the above mentioned objectives are met efficiently and effectively.

The TenStep Project Management Process Phases

The ten phases of the TenStep Management Process are below. The first two fall under planning while the rest fall under the domain of management.

Plan

- Work definition.
- Build budget and schedule.

Manage

- Schedule and budget management.
- Manage issues.
- Manage scope.
- Manage communication.
- Manage risk.
- Manage human resources.
- Manage quality and metrics.
- Manage procurement.

Work Definition

It is essential for the project manager and project sponsor to understand and comprehend the needs and requirements of the project prior to starting the actual work. This will avoid confusion and misunderstandings among them as they will have a common goal to work towards. The project manager clearly defines all the work that needs to be performed during the project management life cycle. Questions like what to develop, how to develop

and when to develop are answered by the project manager. Cost, risk and time required for the completion of the project are determined with the help of project team (TenStep, 2011).

Before defining the work and the plan to complete the work, it should be certain that the requirements provided by the client are realistic and attainable. Planning of the project should be done according to the size of the project and setting shorter deadlines for lengthy projects is likely to put unneeded pressure on team members resulting in errors and mistakes. Therefore, for project success it is essential that appropriate time is spent on planning and work definition (1.0 Define the Work, n.d.).

For larger projects, defining the work can play a very important role in the delivery of the project. It is hard to manage multiple and complex activities involved in a large project, but by mapping out the work the project manager can make the task of achieving goals easier. This will make the project team aware of what is expected of them as well as how and when to complete and deliver the work. Work definition also makes the task of management easier. Every detail of defining the work is recorded in the Project Definition document and it is updated regularly throughout the delivery of the project.

Defining the work will provide the benefits listed below (1.0 Define the Work, n.d.).

- Understanding and approval of project scope, deliverables, objectives, cost, risk and approach.
- Business case validation approval.
- Availability of resources needed for project delivery.
- Creating milestones and providing a clear roadmap to work towards.
- Project management process selection and approval by the client and team members.

Building Budget and Schedule

In this step the project budget and schedule are created. The project manager along with the other team members all meet to input on and complete this effort together. Small projects usually do not need as much formality, but larger projects should always be analyzed and reviewed thoroughly before setting a budget. Microsoft Project, a spreadsheet, or even a simple piece of paper can be used for documenting the budget (TenStep, 2011). Various

techniques and approaches can be utilized in developing the project budget. The selection of the method to employ depends on the nature and size of project. However, it is crucial as it ensures that each team member is aware of their duties and work activities.

The budget is basically the collective summary of expenses that are expected to be incurred during the project life cycle. A detailed description of expenses and their respective costs is provided and the budget can include external costs as well as internal costs depending on the accounting procedures of the organization (2.0 Build the Schedule and Budget, n.d.). Along with the budget, the schedule is also prepared keeping in mind how and what activities need to be performed and by whom.

If a template is not available for budget preparation then the WBS (Work Breakdown Structure) can be used. It divides the activities into sub-activities making the calculation of costs to be incurred easier. Each activity's cost is accumulated to tabulate total cost of the project. TenStep guides the user in formation of the budget from scratch. Each detail is defined clearly and guidelines are available to aid users (TenStep, 2011).

TenStep has tied budget and schedule together even though both are different in nature. The budget shows the cost involved while the schedule shows the activities and their sequence involved in creation of project deliverables. They are tied together because both are fundamental components of the project's success. Their relationship is further detailed with the points below (2.0 Build the Schedule and Budget, n.d.).

- The project will most likely be over budget if the project is behind schedule.
- If the amount of work is underestimated, the budget and schedule will have errors.
- Deadlines may be missed due to the project being over budget and behind schedule.
- Human resources will have a direct impact on budget and schedule.
- The budget and the schedule are part of the triple constraints of project management: scope, resources (budget), and time (schedule). If the scope is increased or decreased the budget and/or schedule will be effected as well.

Schedule and Budget Management

Management of the schedule and budget is as crucial as their development. Once the schedule and costs are finalized, it is the responsibility of the project manager to make sure that the project is completed on time and within budget. Effective management will result in successful completion of the project (3.0 Manage the Schedule and Budget, n.d.).

Prior to this phase of the TenStep life cycle, project definition, project schedule, and the project budget are prepared in step 1 and step 2. The schedule should always be up-to-date to report the current status of the project and it should continue to be updated at regular intervals. Usually schedules are updated weekly. The tasks completed along with the incomplete tasks are identified and updated. Additionally, an evaluation of the schedule is completed to see whether the project will be completed on time or not (TenStep, 2011).

The schedule should be flexible enough to support changes that may occur during the project and after every change the schedule should be updated accordingly. Managing budget and schedule effectively is one of the most important jobs of the project manager because any mismanagement could have major consequences. For example, if the project manager records $10,000 for a budget item instead of $100,000 then this will adversely affect project resources. To avoid such errors the project manager has to be proactive all the time.

The project could be on schedule one week but fall behind schedule the next week due unforeseen circumstances. To avoid this, the project manager and his team evaluate all the possible events that could affect the project and plan strategies to avoid them. These schedule impacts are evident, for example, if an activity on the critical path falls behind schedule and risks causing the whole project to slip.

During times like this, the project manager is not in a position to relax. Instead he/she should evaluate all available resources and ways to get the project back on track. It should certainly be obvious at this point that a project manager must be able to effectively manage budget and schedule to avoid disastrous failures (3.0 Manage the Schedule and Budget, n.d.).

Manage Issues

Minor problems are common during the execution of a complex project. However, many times an ‘issue’ is a problem that cannot be resolved by the project manager and the team members alone. Outside help is required to resolve the problem quickly and effectively.

Initially the issue is documented and evaluated by the whole team and if no consensus for resolution is reached, outside help is sought (4.0 Manage Issues, n.d.).

Issues need to be resolved immediately and they cannot be ignored or put off until the last minute. If they are not resolved as soon as possible they might turn into monstrous never ending problems that may seriously threaten the success of the project. Managing issues is a skill every project manager needs to master because when it comes to finding solutions, the project manager is the person everybody turns to (TenStep, 2011).

There are number of different techniques and methods for identifying and resolving issues. An appropriate method should be selected and employed according to nature of the problem. Any changes required in the documentation should be made and concerned stakeholders should be informed of the decision.

Manage Scope

Change is the only thing constant in this world. The TenStep process realizes this and allows changes throughout the project. This is the reason why project definition and project scope are not finalized in step 1 and step 2. They are initiated to make room for changes that may occur later in the project.

Scope is the term used to describe the boundaries of the project and the total work involved and it defines what the project will or will not deliver (5.0 Manage Scope, n.d.). If we look at the reasons for project failure, two main reasons reveal themselves. First, the work is not defined clearly and correctly. Usually the team doesn't spend enough time in planning the project and the work is defined in a haphazard manner making the goals and objectives unclear. The second reason is scope management failure. This means even if the scope is defined properly, managing it can be difficult and failure can occur.

Without a clear definition of project scope it is close to impossible to manage it. That is why project scope and project definition are identified and described in step 1 and step 2. Once the project manager is aware of the project requirements and what the project is intended to deliver he/she will be able to manage the resources and team effectively and efficiently. The project manager makes sure that the project definition is viable and approved by the stakeholders (TenStep, 2011). Any changes occurring during the project should be documented and the project definition and project charter should be updated in order to keep the scope up-to-date.

Manage Communication

To ensure success of the project it is crucial to have a clearly defined, reciprocal communication path. The team members, project manager, and stakeholders should be able to communicate with each other openly and freely to avoid the element of surprise. Most of the time when a project fails it is not because the requirements were not met but because they were not communicated properly. Either the manager did not understand them or the client did not define them properly. Either way, the project fails resulting in loss of time and money. To avoid such failure it is essential to manage the flow of communication. In fact, this is one of the factors that contribute the most to project success (TenStep, 2011).

There are normally two key forms of communication. They are Status Meetings and Status Reports. Meetings are conducted at the end of each phase of the project delivery life cycle to ensure that the team is on the right path and to let stakeholders know about the progress of the project and get their feedback. After this meeting, and at other times as necessary, status reports are prepared documenting changes made, which are then processed and approved by the client and stakeholders (6.0 Manage Communication, n.d.). Before starting the project the communication paths should be clearly defined. Questions like, who is answerable to whom and who monitors the work progress should be answered and a clear hierarchy should be defined to make the project easy and smooth.

Manage Risk

Risks are potential future conditions that will affect the project adversely if they occur. Issues, on the other hand are current problems that should be dealt with immediately. Risks can also be called potential future issues. Eliminating the occurrence of risks is not in the hands of the project manager or delivery team, but precautionary measures can be taken to minimize or avoid them.

It is the responsibility of the project manager to try to foresee the future and identify potential risks, but not all risks can be anticipated. For example, while times of high risk in certain geographical areas during specific times of the year can be identified, the occurrence of a natural disaster cannot be ultimately foreseen. To manage risk, the project manager should first identify them. After the identification, precautionary measures are taken to mitigate or avoid the risk. High, low, and medium level risks are identified. This is the advantage of risk management and it strives to eliminate the chances of project failure.

Small projects can be much easier to manage since their duration is not as long, whereas longer projects require complete and thorough risk assessment along with a risk response plan. Throughout the project delivery life cycle the probability of risk should not be ignored, but rather it should be managed in a way that its occurrence is minimized. The project manager is responsible for making sure all risks are being monitored and if a risk becomes an issue, activating the risk management plan (TenStep, 2011). After the assessment of risk, the Risk Management Plan is developed keeping the requirements in mind.

Manage Human Resources

It is the responsibility of the project manager to manage every resource needed for the delivery of the project. Management of team members and every other person involved also falls on the shoulders of the project manager, although he/she shares these responsibilities with the functional managers. Managing a large number of employees can be problematic, however, since everybody will have their own opinion on how to do the work. Effective management is crucial to project success because without a proper management system there would be chaos and confusion among the delivery team.

The project manager evaluates each member of the team before assigning them their respective tasks. Their expertise and skills are taken into consideration and their flaws are documented as well. Appointing a person appropriate for the work makes management easier, as the person is experienced and is aware of the nature of the work. On the other hand, if an inexperienced person is hired to do the job he could possibly fail miserably taking the whole team down with him.

For these reasons and many others, a project manager who is good with people is likely to be more successful than one who is just good at managing the project.

Manage Quality and Metrics

The goal of the project is to meet the customer requirements. Therefore, it could be correct to say that the customer defines quality and determines whether it has been delivered or not. Most of the time quality is associated with using the best material, the best team and absolutely no defects. But the client is not looking for a picture perfect solution and most likely couldn’t even afford it.

The aim of quality management is to first identify and evaluate customer requirements and then formulate a plan accordingly to meet those requirements. One of the major roles of quality management is to identify bugs and errors in the early stages of the delivery of the project because identification and resolution at a later time is more costly and time consuming. Quality management should begin prior to the start of the project. It is a time consuming process as it involves every aspect of the project deliverable, the team involved, and the resources to be used. Each and every thing is scrutinized before finalization. It may take a lot of time at the beginning of the project, but this time will certainly pay off later (TenStep, 2011).

Every deliverable at the end of each phase has to pass quality tests before it is delivered to the client. If it does not meet quality standards, the deliverable is modified until approved by the client. While the client ultimately sets the quality standards, the project manager facilitates communication to the team and breaks them down into simple terms while focusing on the one that is most important for the client. For example, the client's main focus may be on error minimization rather than 20 different functionalities. Small projects do not require a lengthy quality management plan whereas for medium and large projects it is very crucial to have a fully defined quality management plan.

Quality and metrics go hand in hand. Managing metrics is as important as managing the quality of the product. Metrics are basically used to measure the quality from start to finish identifying trends in increasing or decreasing quality of the project. Improving the quality of the process and work can be very difficult if the process is not being documented. Besides being the basis of quality management, metrics can also be used to evaluate the success of the project (9.0 Manage Quality and Metrics, n.d.).

Large projects, especially, should have clearly defined metrics as it also provides the team a common goal to work towards. Before defining quality criteria the requirements of every stakeholder associated with the project should be taken into account.

Manage Procurement

Procurement refers to obtaining supplies and resources from outside the company. This specifically refers to outside vendors and suppliers, not to the ones within the organization. Every project manager needs to understand the procurement process at some point during project delivery because this is yet another area where the project manager gives

his input. That said this is often one area within project delivery where the project manager does not have any authority. Many times, he doesn't have the power to sign contracts on behalf of the company nor does he administer the contracts being signed.

If procurements are needed for the project delivery process a procurement plan is formed based on the procurement contracts signed by the organization. For example, hardware will be purchased using the standard company contract. Similarly it will be purchased from the vendors approved by the company. The project manager usually does not have any say in these matters.

The selection of suppliers is often considered to be a part of the project delivery process. Therefore the planning is done in the initial Analysis Phase after project execution has started. However, that said, this activity may sometimes be performed as part of the up-front project definition process (10.0 Manage Procurement, n.d.).

TenStep Process Summary

- **Initiate the Project:** The TenStep process helps in accurate planning of the project. The initial documentation is prepared and approved and every organization has a process to govern the project with the characteristics below.
    - Identification of all possible projects.
    - Narrow down and pick the processes most suitable for the project and in alignment with the goals and objectives of the company.
    - Document everything (i.e. Business Case, etc.).
    - Identify the most important requirements that need to be met.
    - This step is to initiate the project, gather the requirements and get approvals. Initiation doesn't have to be the starting of the project. The project may be started quite later after it has been initiated (TenStep, 2011).
- **Plan the Project:** As described earlier, planning of the project is very crucial. Once the need to execute a project has been identified, it is now important to identify the resources needed, the risks associated with the project, the deliverables of the project, the stakeholders, and the delivery process. All the decisions are formulated in the planning phase. The project manager, along with

the stakeholders, make these important decisions and plans in every aspect of the project. A full plan from start to finish is formulated including the tasks that need to be performed, the duties of the workforce, the tasks list, and sub division of the tasks. If the project is not planned before starting it, the team will not be aware of what is expected of them and what their actual goal is. This will lead to project failure. Also, the project manager should evaluate the success probability of the project accurately before starting. If the project is not likely to succeed, it should either be terminated or rethought (TenStep, 2011).

- **Execute Project Management on Your Project:** After detailed planning of the project is completed, it is executed. Execution is the actual delivery of the project. The tasks assigned to the team members are performed while the project manager keeps an eye on them. If an issue arises, it is the responsibility of the project manager to resolve it through issue management. This phase is further divided into two parts: executing and monitor and control (TenStep, 2011).

- **Monitor and Control the Project:** This process is more active and goes on throughout the project delivery of the project. After the plan has been developed and execution has started the project manager monitors the project and tracks its performance against the plan. It is the responsibility of the project manager to keep track of all activities so that if something goes wrong he will be able to handle it effectively. For example, if the scope changes during the project delivery phase, then the project manager should be capable enough to adjust the whole project according to the changes. This is done through scope management (TenStep, 2011).

- **Close the Project:** The closure of the project is as important as the start. It should be closed properly by making sure all the requirements of the client have been fulfilled and the final product performs the desired functions. After the project is completed a full report is generated including the success and failures of the whole project. The good and bad are all documented for future use and the project delivery team is disbanded. Rewards or punishments are also given after the project is completed. Even if the project is unsuccessful and does not meet the requirements of the client, these activities are performed to capture lessons learned for future teams. The activities below are performed at closure.

- Hold project postmortem meeting.
- Success of failure declaration.
- Turn over project files.
- Conduct performance reviews.
- Reassign project team members.

For the successful completion of the project it is important to follow all the rules and regulations described by the TenStep Project Management Process. If they are not followed clearly the chances of project failure will increase.

Advantages of TenStep

Listed below are advantages of TenStep.

- Easy to follow because of step-by-step approach.
- Project Manager can decide on the complexity of model as per the project need.
- Flexible as well as scalable.
- Provides ease of implementation as well as understanding.
- Crafts a proper roadmap for effective implementation.
- Suggests proper ways to manage risk and quality.

Disadvantages of TenStep

Listed below are disadvantages of TenStep.

- Steps include former steps so skipping one step can be an issue.
- Proper closure of the project is necessary.

Chapter 6

Agile: Extreme Programming (XP)

Agile Development is a term for a group of iterative and incremental software development methodologies, of which the most popular are Extreme Programming (XP), Scrum, Crystal, Dynamic Systems Development Method (DSDM), Lean Development, and Feature-Driven Development (FDD). We will cover these in chapters of this book beginning here with Extreme Programming. Extreme Programming is a software development process that is part of Agile development. Software is developed in increments and customer satisfaction is the primary objective. Extreme Programming is a combination of straightforward practices based on common sense and stressing the importance of communication, teamwork, and client satisfaction. It supports the values that are mentioned below (Marc Clifton).

- **Communication:** successful communication helps produce and deliver a software project on time. There should be constant and thorough communication between members of the project team.
- **Feedback:** Client involvement and feedback are essential measures of client satisfaction.
- **Simplicity:** XP stresses the necessity to make things as straightforward and as simple as possible. The project should have attainable goals and objectives and should produce the desired output thereby satisfying the customer.
- **Courage:** Developers should have the courage and confidence to make changes and develop quality products.

History of Extreme Programming (XP)

Extreme Programming was designed by Kent Beck during his work on the Chrysler Comprehensive Compensation system (C3) payroll project. Beck became the C3 project manager in March 1996 and began to modify the development methodology used in the project and subsequently wrote a book on this methodology (in October 1999, *Extreme Programming Explained* was published). The C3 project was cancelled in February 2000

after 7 years for undisclosed reasons by Chrysler, when the company was acquired by Daimler-Benz.

Although extreme programming itself is relatively new, many of its practices have been around for some time. The methodology, after all, takes “best practices” to extreme levels. For example, the "practice of test-first development, planning and writing tests before each micro-increment," was used as early as NASA's Mercury Project in the early 1960s (Larman 2003). To shorten the total development time, some formal test documents, such as acceptance test, are developed in parallel, or shortly before, the software is ready for testing. A NASA independent test group can write the test procedures based on formal requirements and logical limits before the software has even been written and integrated with the hardware. In XP this concept is taken to the extreme level by writing automated tests, perhaps inside of software modules, which authenticate the operation of even small sections of software cod, rather than only testing the larger features.

Core Practices of XP

1. **The Design Game:** Business and development together provide the best price as well as the best chance at success. The design game takes place at different phases, but the basic rules are the same.

    - Business comes up with the requirements for the project. Each requirement is then documented as a User Story that describes in detail what's needed? User stories are generally written on 4x6 cards.

    - The development team estimates the effort that is needed on each User Story and how the team will accomplish the desired goal.

    - Business then decides what stories to implement and in what order and how and when to provide incremental outputs from the project.

2. **Small Releases:** XP groups ensure that each release in the life cycle is small because requirements often change. Increments are kept small so that client’s requirements are fulfilled efficiently and effectively throughout the project. Small releases reduce the chance of failure because the client will be able to identify errors and request a remedy immediately. If the release is not able to provide business value to the client, the project may be terminated altogether. Short

releases also help the delivery team deal with changes effectively and plan accordingly (Martel, 2002).

3. **Simple Design:** One of the most important assumptions made in extreme programming is that requirements are ever changing and may in fact change tomorrow thus the focus is on doing solely what is required to satisfy today's requirements. XP uses the most effective process that gets the job done. Due to these unpredictable changes the budget for the project is also always uncertain. It has the tendency to fluctuate as the project progresses. In XP, the design delivered is the easiest to understand, has the least amount of classes and methods, has no code redundancies, and passes all tests. XP doesn't believe in investing in the future. The sole focus is on ensuring that today's work is on the mark. Keeping the design as simple as possible helps the team to work effectively and avoids unnecessary errors. There is less paperwork and the team can focus on the project development itself (Martel, 2002).

4. **Metaphor:** A metaphor is a clear statement that represents the technical as well as business aspects of the project. It is like a scope statement that defines clearly what the project hopes to achieve and how it should be done. The metaphor is sometimes incorporated into a single user story to give everyone a basic idea of what, how, and when to do tasks. It can also be defined as a high-level architecture for software development (Martel, 2002). At its best, the figure of speech (metaphor) could also be described as a straightforward evocative description of how the program works and what it hopes to achieve.

5. **Continuous Testing:** Software testing is the key element of Extreme Programming (XP). Automated regression testing is the main form of testing done in XP. The client defines the requirements (acceptance testing) that drive how the team develops the software. These tests also verify whether the project provides the required functionality or not. In XP, unit tests are written before the programmers write the actual code. This helps the programmer think about the problem before writing the code that provides the solution. The tests are used as raw material input to construct functionalities of the software. Once a function has been developed completely, these test cases are used to evaluate whether it performs the desired function or not. Every increment is tested before delivering it

to the client. After the increment has been delivered to the client acceptance testing is done to confirm that the required functionality has been provided.

6. **Refactoring:** Refactoring means changing the code over time to accommodate the client's changes without changing the functionality of the software. The reason XP supports simple and understandable code is because there are not many documents to describe it. So the code should be written in a way that makes future changes possible and keep its maintainability intact. Once the code has been written the developer should check it to make sure it is as simple as possible. In case it isn't then it should be reconstructed (refactoring). Automated test procedures are used as a basis to incorporate changes. If the changes are accepted then they become part of the next baseline and if they are not accepted, the programmer can change them right away. Constant refactoring ensures that the design is as simple and easy to understand as possible (Martel, 2002).

7. **Pair Programming:** XP promotes the concept of paired or combined programming. It means that two people (programmers) write the code together while working on one machine. One person controls the mouse and keyboard and ensure that the approach used for writing the code is simplified enough to be understood by anyone. The second person, on the other hand, thinks strategically and decides if the chosen approach is suitable for implementation of the function or not. Both programmers work together to develop the code. The code has to be simple, but still satisfy all the requirements of the client and the pair keeps switching roles throughout the development process. Project managers often object to pair programming on the bases that it increases the overall budget of the project. However, in most cases, the increase is not more than 15% and pair programming increases customer satisfaction ten times more than the cost. Due to the increased customer satisfaction the increased cost is usually overlooked. However, in situations where pair programming is not acceptable, the project manager and the development team might replace it with software inspection and measurements (Martel, 2002).

8. **Collective Ownership:** In XP each team member can make additions to the code. It is collectively owned by them and they are required to make any improvements they deem necessary. No single person "owns" a module. Any developer is

expected to be ready to work on any a part of the code base at any time. The advantage is that the programmers can get instant feedback to see whether their idea works or not. Also the automated tests let the programmers modify the code more freely and without the fear of failure. (Martel, 2002)

9. **Continuous Integration:** All changes are incorporated into the code base at least daily. This ensures that there is always a deliverable version available for the client containing all new features and that there is a current baseline for future incremental releases. Infrequent integration results in serious issues for the software development team. If changes are not integrated regularly, the team may miss a very important requirement that could lead to disappointment on the customer's end. The best practice for continuous integration is the use of pair programming. Each programmer should take turns integrating his/her code to see how the program works. If the program fails the test the changes can be undone to make room for new changes (Martel, 2002).

10. **On-Site-Customer:** The main aim of XP is to satisfy the customer. Therefore, customer feedback is really important for software delivery. The initial requirements are documented and user stories are formed but as discussed earlier, the requirements have the tendency to change. So when the developer incorporates these changes, customer feedback is essential. For this reason, one person from the client's end is included in the team. He represents the interests of the client and provides feedback related to any changes made. This helps in enhancing business value as the team will be able to contact the customer instantly in case a problem arises. This also makes it easy for customers to add requirements on short notice.

11. **Sustainable Development:** No team member is allowed to work more than 60-hours per week. Working more than that is believed to reduce productivity thus reducing the product quality. Also if the team members are forced to work more than the assigned hours then it may lead to high turnover rates and increase in human errors. XP emphasizes that team members should be fresh and active while working and overtime can make them lazy and dull (Martel, 2002).

12. **Coding Standards:** since the program is developed in collaboration with different teams it is important to have a predefined coding standard for all programmers to

use. This ensures that the code is developed in a similar way that is easy to comprehend. Also programmers will be able to understand and edit the code of other programmers working on the project. Additionally, clearly stated standards also provide the developers a common goal to work towards. The coding standard should be easy to follow and should have the ability to support changes.

These 12 XP core practices are applied while software is developed using Agile Extreme Programming. Of course not all of them can be applied at the same time since they may have conflicting interests or it just may not be possible. Adopting even a few of the main practices will help a project achieve its technical and business value. Agile Extreme Programming is designed to increase the efficiency of the development process and to satisfy the customer. It is more flexible and adaptive to changes making it different from other development processes.

The Values of Extreme Programming

Extreme Programming is based on a set of values. XP emphasizes the importance of meeting corporate goals as well as maximizing the value of the product. There are four main values that are followed while using XP, however, there is no hard and fast rule about implementing just these four values. Additions can be made because XP is flexible enough to accept changes. The four values are listed below.

- Simplicity.
- Communication.
- Courage.
- Feedback.

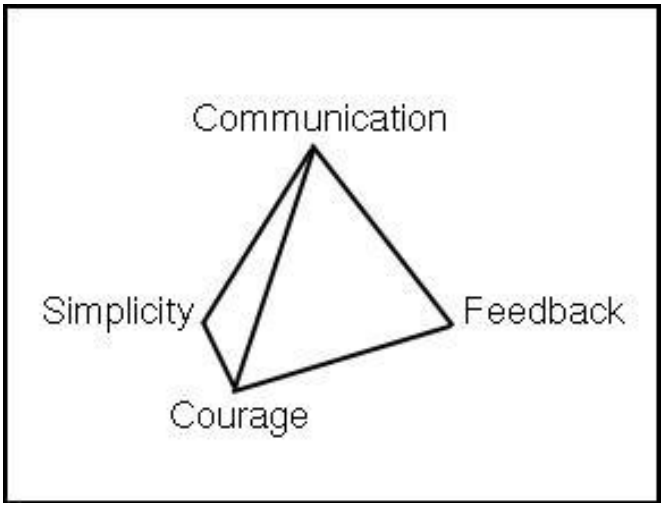


Figure 6.1 Extreme Programming (XP) Values

Simplicity: The requirements and specifications are kept as simple and straightforward as possible. A simple design is easy to implement and consumes less time as compared to a complex design. Also it saves money and the team can easily comprehend it. It is always a good idea to choose a simple design over a complex one for the benefit of project success. Simplicity of the design is decided based on the nature of the project. That basically means simple for one team can be complex for another, so when deciding what and how to design the project, the requirements are taken into account along with the resources available. Each team member gives their opinion on how the code should be written. It is a subjective task because it can vary according to the opinion of the team members and project requirements. The final code design should have the four subjective qualities listed below (Wells, 1999, 2009).

- **Understandable:** The design should be one that is understandable to everyone. Even the new recruits should be able to understand it on their own or with little guidance. Complicated designs will make the system complex and increase the chances of error.

- **Testable:** Unit tests and acceptance tests should be written to automatically check for problems. This affects the overall design of the system. The system should be broken down into smaller testable units as this makes problem identification mush easier.

- **Explainable:** Extreme Programming encourages simple design so that it is easy to explain should the need arise. The customer or other team members might want to

understand the code and if the developer uses a complex code design then he will have difficulty explaining it to others.

- **Browsable:** This is the ability to find what you need when you need it easily. While writing code, variable names that are easy to find and remember should be used. Correct use of Polymorphism, Inheritance and Delegation makes it easier to find items.

Code can only be written in a simple way if the developer has the knowledge to recognize a simple design. A person can have plenty of information but it is not always knowledge. Knowledge is the ability to be aware of the project domain and the problems associated with it. The best approach is to build a simple design for only the functionality you are about to implement. Keeping things as simple as possible will surely help avoid failure, however, keeping it simple is not an easy job (Wells, 1999, 2009).

Communication: The importance of communication should not be understated. It is one of the vital values of Extreme Programming (XP). Successful communication ensures project success especially when it comes to software development because lack of proper communication among the team members could lead to major errors and faults.

Everyone is part of the team so face-to-face communication is vital. Since XP doesn't place importance on documents the details and requirements have to be communicated orally. The main emphasis of XP is on communication rather than tedious paper work. Face-to-face communication is still the best mode of communication. Not only is the team able to understand the requirements but they are able to ask questions to avoid ambiguities and also see the body language of team members, which is not possible through documentation. Team members need to be cautious about what they say and how they say it because every person has a different perception of a situation and if the whole team is not on the same page when it comes to requirements then it could lead to disastrous results. Additionally, those that are the receivers of information must be attentive during the meetings because any distractions could lead to misunderstanding of the client's requirements. This is a common error that usually leads to disappointment on the client's end (Ropa, 2011).

Now, if we look at things rationally, we see that face-to-face communication can be very effective within the team but what about the stakeholders and customers? How do we communicate each requirement with them? In answer to these questions Agile Extreme

Programming (XP) encourages the team to put together a large document containing the initial requirements, specification document, initial design, and some code for software development. This document is forwarded to the stakeholders, customers, and the delivery team. Throughout the development process multiple meetings are held to review the project plan further. This promotes better written communication as well as oral communication.

The code written in these documents should be simple, as mentioned earlier. If complex, hard-to-understand code is included in them then the act of communication will suffer greatly. The code written is not final. It is changed throughout the development process. During review meetings, comments are added to the code to make easier to understand. In Agile development automated acceptance tests are developed as part of the communication process. This works perfectly since documentation is just a formality (Ropa, 2011).

Courage: Fear is a part of everything. While developing software a team has to face many fears. The fear of not meeting deadlines, the fear of not satisfying the customer, the fear of a budget shortage, and on and on. To face these fears bravely the team needs to have courage. Every member should be courageous enough to know they can handle any unforeseen situation correctly. Courage is required at all levels. From the project manager to every person involved in the delivery of the project, everyone should be courageous. XP promotes open communication, which means that no team member will have to face their fears alone. Each member is obliged to respect and support every other member of the team. Respecting each other's opinions and talents creates harmony in the team and it also promotes a healthy and friendly environment to work in. Every team member is human, so naturally if they fail or error they will feel as if they have let down the team. At times like these the team should stick together and show courage to face every difficulty that presents itself during the delivery of the project (Extreme Programming Model, n.d.).

Too much of anything can be bad. As such, too much courage can be dangerous as well. The XP methodology strives to use small dosages of courage to ensure success without getting out of hand.

Feedback: Feedback is essential to monitor a project's progress and the more immediate the feedback is, the better the results. Continuous communication throughout the project makes getting feedback easy. The client can request changes and it is the responsibility of the team to deliver them within the given time frame and get feedback. If

deliverables are not up to the expectations of the client then further changes can be made. If the client's feedback is not received, then the team will never really be able to evaluate the success of the project.

Feedback starts when a programmer develops the initial incremental software release for the client. The client then uses it and provides feedback identifying any defects. This feedback is then used as the basis for next the incremental release. The identified defects are corrected and any changes requested by the client are incorporated (The Four XP Values, n.d.).

Advantages of Extreme Programming

Extreme Programming is different from traditional approaches to programming using other methodologies of the past. The advantages of extreme programming are listed below (Nayab, 2010).

- **Robustness:** XP promotes simplicity. The design is like a jigsaw puzzle where different iterative parts are developed simultaneously and are assembled at the end of the project. This ensures that the software is developed faster with a minimum of defects. Frequent automated tests reduce defects and user acceptance testing identifies any other defects undetected by previous tests. Also XP allows cost-based software development. This allows the customer to decide what to include and what not to include accordingly. Since the budget is predefined, the customer can pick the most important functionality to be implemented first. This way the customer will be able to obtain maximum value to increase the business worth of the project. This also allows the user and delivery team to shut down the project anytime they want. However, if it is shut down the essential functionalities will have already been delivered.

- **Resilience:** One of the success criteria of a project is that all requirements are delivered. However, frequently this is not the case with most projects because of scope changes. Changes can occur for many reasons such as changes in the business environment or technology. XP has the flexibility to accommodate changes and it is designed in such a way that any changes made by the client later in the project can be accommodated in an incremental release. These changes are typically the result of customer feedback.

- **Cost Savings:** Extreme programming has the ability to eliminate any unwanted and unnecessary functionality thus saving cost. It allows the developer to focus solely on code design rather than paper work. This saves time as well as money. The highest costs involved with Extreme Programming are the costs of changes. However, since changes are generally made constantly throughout the process, the cost is high. Changes are usually made more than 5-100 times in a project. Compared to conventional programming techniques, XP is much more accommodating of changes made in the development phase of a project.

- **Less Risk:** One major advantage of XP is that it mitigates risk. In conventional programming the responsibility to write the code is solely on the programmer but when it comes to XP the code writing is distributed among different team members to overcome risk as well as complete the task quickly and efficiently. Also, due to continuous communication among the team members and with the client the chances of project failure are also reduced.

- **Employee Satisfaction:** The main objective of XP is to satisfy the client. Client's requirements are made the baseline for code development ensuring that the software provides the desired functionality. The breaking down of tasks into iterative releases satisfies the client that work is being completed and his changes in the requirements are followed. Also, iterative releases help ensure that the project is completed before the deadline (Nayab, 2010).

Disadvantages of Extreme Programming

Listed below are disadvantages of Extreme Programming (Dip, 2009).

- The major disadvantage of XP is that it assumes continuous involvement of the client throughout the development process. However, in reality the client may not be available all the time. They have other duties to perform and other tasks to take care of. The customer himself is not present on the site the whole time, but rather has a representative present to keep an eye on the project.

- Another disadvantage is that it lacks proper documentation. The main focus of XP is on code development rather than on documentation. If the details of the project needed to be reviewed in the future there would be very few documents available to consult. The whole code would have to be reviewed over again.

- XP does not believe in the idea of static requirements and cost. They fluctuate continuously throughout the project life cycle. Other programming techniques, on the other hand, encourage detailed static planning from the start.

- Pair programming can lead to a large about of duplicate code that makes unit testing a hectic job. This eventually will take a lot of time to be tested completely and some code might be left out.

- XP does not measure the project in terms of quality. The main focus here is customer satisfaction and every necessary step is taken to achieve that irrespective of the quality.

- Another disadvantage is that software code can be very large and a bit twisted in nature. It may be easy for the delivery team to understand it, but for the client it could be a problem.

- XP encourages changes, however, too many changes can lead to the loss of the main function of the product and also waste time.

Chapter 7

Agile: SCRUM

Advancement in technology has led to the increase in the complexity of project requirements. Additionally, via change control, typically a large number of complex requirements are added to a project during delivery. Project failure, though, is not in the complexity of the requirements; rather it is in the management of the requirements that poses a threat to the success of a project. As mentioned, to resolve problems like these, Agile was developed. It delivers the project in small increments hence reducing the chances of failure immensely.

SCRUM is an Agile development process focusing on iterative and incremental software development. It is an approach used in top industries with a very high success rate. It focuses on customer feedback making it an instant hit from the very beginning (Cohn, n.d.).

The name SCRUM has originated from the word “Scrimmage” used quite frequently in rugby. It signifies players wrestling with each other on the field to get the job done. SCRUM, unlike other Agile development processes, incorporates the aspects of both developmental and managerial operations. It follows the sequence of prototyping in that it not only supports changes in the beginning, but it encourages changes throughout the development process (serena.com, 2007).

There are three basic principles that SCRUM is based on: visible progress, constant inspection, and adaption. Using these three principles, SCRUM provides a very useful and convenient environment for software development. The delivery team focuses on the changing requirements and incorporates them as the project progresses towards completion. The team is generally highly motivated because they get to choose the project they want to work on and how they will be performing their respective tasks. The team members get to put their heads together to deliver fully functional, small, incremental releases of the software to the client. Any changes wanted by the client are documented and incorporated in the next incremental release. Each team member has a specific name and a role to perform. This will be discussed later in this chapter (Correa, n.d.).

SCRUM has a rather simple framework based on common sense principles and is easy to understand and adopt. It doesn't use highly complicated engineering terms to make the process complex. It follows an empirical approach in that the team makes decisions based on their experiences rather than based on some instructional manual provided by management. This creates a light and easy atmosphere for the team to work in. They are often satisfied with the work they complete and strive to achieve better result over time (Correa, n.d.).

History of SCRUM

SCRUM has gained popularity ever since its inception. Interestingly, though, sources reveal that the exact people to create SCRUM are not certain. Industry professionals are split between two groups as to who created SCRUM. Some believe that SCRUM came into existence in 1993 and the creators were John Scumniotales, Jeff McKenna and Jeff Sutherland, while others believe Ikujiro Nonaka and Hirotaka Takeuchi created SCRUM in 1986 (Krishnamurthy, 2012).

Additionally, it also can be found from various sources that Ken Schwaber and Mike Beedle created SCRUM. According to a white paper written by Beedle, he states that he learned the SCRUM method from Jeff Sutherland and then later introduced it to Ken Schwaber (Krishnamurthy, 2012).

By looking at different sources, though, one can ultimately conclude that the Agile development method that is SCRUM was in fact developed by Hirotaka Takeuchi and Ikujiro Nonaka. They came up with the initial concept that was later reformed by other professionals. Sutherland modified the concept and was the first person to apply it. He along with Schwaber presented their experience at the OOPSLA conference in 1995. They also wrote a book named "Agile Software Development with Scrum" to enlighten people about this method and how it can be useful for their projects (Krishnamurthy, 2012).

Since then SCRUM has been in use and has resulted in many successful projects worldwide. Big companies like Google, Microsoft, Yahoo!, Motorola, GE, Cisco, SAP, Lockheed Marin, the US Federal Government, and Capital One are using SCRUM for their projects and have reported high success rates. However, the use of SCRUM is not limited to big projects; small businesses around the globe are also utilizing SCRUM for better results. Companies using SCRUM have reported to have better productivity, increased motivation

among their employees, timely completion of projects, and enhanced reputation of the company thereby increasing SCRUM's credibility (Benefield, 2007).

The Agile development process has always been popular among developers compared to traditional development methods. It allows developers to get instant client feedback and alter the next incremental software release accordingly. It saves money as well as time and it favors small cross-functional teams that work together to achieve results versus massive hierarchies. More and more organizations are now opting for Agile development, especially SCRUM, because of its high success rates. SCRUM has defined all the approaches involved in delivery of a project and everything is not only in order, but also contains all the essential ingredients. Interestingly, it has been shown that SCRUM is not only for software development, but can also work wonders for other industries (Schwaber).

SCRUM Roles

SCRUM consists of three major roles that drive a project. They perform their respective roles effectively for the betterment of the whole project. The primary roles are The Product Owner, The Team, and The ScrumMaster. Each of these is discussed further below along with their respective responsibilities (Benefield, 2007). Additionally, SCRUM focuses on the factors immediately below in forming a team for any project.

- **Focus:** The team is obligated to focus on the task at hand and should not be worried about past projects or potential future projects.
- **Courage:** The team as a whole is a source of support and courage for each other and they have the means to carry out the required tasks, which should give them courage to work effectively.
- **Openness:** It is essential to have an open environment for communication while dealing with each other. Openness among team members promotes friendliness and avoids conflicts.
- **Commitment:** The team members should be fully committed to their jobs and thus to their individual and organizational success.
- **Respect:** Without respect for each other no team can survive the challenges faced while working together. The absence of respect will promote hatred and hostility

towards each other and therefore, respecting each other and their opinion is crucial for the success of the project.

The Product Owner: The Product Owner is the backbone of the project and is responsible for its successful completion. He is the key stakeholder that coveys the vision of the project to rest of the team. It is also the responsibility of the Product Owner to outline the SCRUM Backlog and ensure it reflects the business value. The SCRUM Backlog is the list of all the features of the project. It also includes any changes made by the client and provides a guideline to the team. The Product Owner makes sure the requirements specified in the backlog are in accordance to the business image and reflect the nature of the client's demand. Additionally, the product owner has to be present all the time to answer any queries the team might have. It is also his responsibility to make sure the project scope is within the time, budget and resource constraints of the effort. Stakeholders are very important to any project's success and, as such, the product owner is responsible to include them in important decisions and work with their consent (Methodology, 2013).

The product owner, as a leader, should be knowledgeable about the market place to predict the success of the project as well the competition and the domain the team is working within. This varies according to the nature of the software product being built. If the product is to be used internally then it will have different requirements compared to products developed for external use. He should continuously motivate the team and encourage them to give their best at all times (Topics in Scrum, 2012).

The PO (Product Owner) conducts a Sprint Planning Meeting and a collection of tasks that can be completed in each sprint are identified. The most important tasks to be completed are included in one sprint and the rest of them are left for later sprints. This meeting is conducted before each sprint to discuss all the possible ways to proceed with the plan and to make a backup plan in case of any unforeseen disastrous situation. The PO prioritizes the tasks according to their nature and then assigns them to the developing team while keeping an eye on them and guiding them as needed. The PO doesn't have the authority to change the sprint while the work is in progress. He has to wait for the next sprint meeting to initiate the changes. Changes while work is in progress are against SCRUM policies because it creates confusion among the team members (Methodology, 2013).

The PO has the most important and critical job. At times the PO and the client is the same person, but other times there can be more than one client and the project cannot have

more than one PO. The product owner has the decision power and will be held responsible it the project fails. Therefore, the PO has to be really cautious while making decisions. The most important factors should be considered in all decisions, while the insignificant ones should be ignored. The PO has to aggressively interactive with the team and should constantly be the means of motivation and encouragement. He should be fully committed to the project and take full responsibility of all actions (Cohn, n.d.).

The Team: In SCRUM development methodology the team consist of 7-8 members. The team is made up of different teams from around the organization promoting the cross-functional culture. Skillsets on a SCRUM team include a Software Engineer, QA Expert, Programmer, Architectural Engineer, UI designer, and others as required. Cross-functional teams are generally the best for any project because they help reduce group think and provide a higher quality result. The team stays in one room while working on the project and it is known as the "Team Room". This promotes easy communication and accessibility at all times via colocation. The product owner fully cooperates with the team and is available at all times for them (Basics, 2010).

SCRUM is different from other methodologies because in SCRUM the team has the right to decide how much work to take on. They design their own work plan by making user stories and taking on the requirements they find interesting. The product owner doesn't have any say in how the team managers their work as long as they get it done on time. This helps to avoid assignment of excessive workloads to them by the project manager thereby pressuring the team to achieve unrealistic goals. The SCRUM process supports two-way communication. The Product Owner and the team negotiate the work load distribution in the sprint planning meeting (Basics, 2010).

The team works in autonomy and develops their own project schedule and plan. The PO doesn't interfere as long as the work plan is in alignment with the time, budget and resources assigned to the project. The team may complete the tasks within the first half of the sprint or wait until the end of the sprint to complete them. It is totally up to the team to decide as long as they get positive feedback from the client. However, usually the team doesn't have any time to waste because there are always goals to meet and customers to satisfy. As with any project, even if the project is completed on time and within budget there may have been one little detail missed that can result in the rejection of the entire project. So the team must always remain vigilant and concentrate on all details of the project even the minor ones. At

the end of the sprint, the team reviews their work in a Sprint Retrospective Meeting. A Sprint Retrospective Meeting is conducted at the end of each sprint to analyze how well SCRUM is doing for the overall project and what changes are needed to be made. This meeting is conducted in the presence of the team along with the Product Owner and ScrumMaster.

For the team to be successful it is also important for everyone to respect each other and share a mutual understanding of the project. The requirements and goals are clearly communicated to avoid conflicts between the team members. That said, cross-functional teams may be exposed to more conflicts than departmental teams. Largely, this is because people from different organizational departments come together to work on a single project, but may still have bad inter-corporate history as well as different business approaches. . It is the responsibility of the team to realize that such behavior could result in the failure of the project and will ultimately effect the reputation of the organization and the team members. The product owner must also look for negative conflict within the team and be prepared to immediately resolve any issues that arise.

Failure to deliver the project on time will not be blamed on any one single member of the team. It will, in fact, be the responsibility of the team as a whole and they will bear the consequences together. This represents unity and accountability of their actions within the SCRUM team.

ScrumMaster: A ScrumMaster is also one of the critical roles of a project. He should be experienced, mature, and qualified to deal with the everyday challenges faced while working on a project (Alliance, n.d.). A ScrumMaster may also be referred to as “Servant Leader” or an “All Rounder” who helps the team follows the SCRUM process correctly. He should have complete knowledge of SCRUM methodology and how it can be applied to yield maximum results. He needs to be present in the team room at all times to provide guidance and assure no mistakes in the delivery process. The ScrumMaster is also a leader who guides the team in the right direction and leads them to success (Malik A. , n.d.).

A ScrumMaster is not the same as a project manager. A Project manager usually has more control and often does not like delegating authority to subordinates. A ScrumMaster, on the other hand, is a team member who works hard for the success of the team and their work satisfaction is very important to him. A ScrumMaster often delegates decision making authority to the team to make them feel important and to get the work done effectively and efficiently (Admin S. , n.d.).

The ScrumMaster does everything within his power to ensure the team has a comfortable environment to work in and all team requirements are fulfilled. Daily meetings are conducted by the ScrumMaster to discuss the progress of the project and to allow him to intervene with any problem or ambiguity that any team member might have. He is the link between the Product Owner and the team. He makes sure that the product is of high quality and on the route to success. The ScrumMaster can also be viewed as the team protector as he makes sure the team isn't overburdened with excessive work creating a stressful work environment. He ensures the team members are relaxed and ready to work on the tasks assigned to them. This means the ScrumMaster's responsibilities include everything from a minor problem like room temperature to providing backups in case computers crash. To sum it up quickly, we can say that the ScrumMaster's responsibilities are infinite (Admin S. , n.d.).

The ScrumMaster doesn't usually have authority over team members. He does, however, have authority over the process. He cannot fire members, but a ScrumMaster can definitely change the sprint time if he deems it is necessary. The ScrumMaster is more like a coach or a trainer for the team, helping to motivate and guide them throughout the project (Admin S. , n.d.).

A ScrumMaster is also a Quality Manager and a Process Master keeping an eye on the project at all times. He makes sure the quality standards are being met and the process followed was the one agreed on in the sprint planning meeting. The ScrumMaster protects the team from all kinds of external and internal threats. He ensures that the stakeholders and team members are in harmony regarding the requirements and everything is moving as smoothly as planned. Once the project is completed, the ScrumMaster does a performance appraisal of each team member and collects feedback. These are used as reference in future projects and also to improve the overall organizational process (Malik A. , n.d.).

SCRUM Development Phase

The SCRUM development phase consists of the attributes below (What is SCRUM Development?, n.d.).

- **Team Creation:** A creative and goal oriented team is formed consisting of no more than 7-8 members. The team defines the practice, terminologies, implementation strategies, artifacts, and meeting schedules.

- **Backlog Creation:** Three types of backlogs are present in SCRUM. Each of them is later described in this chapter.
    - Product.
    - Release.
    - Sprint.
- **Project Segmentation:** The whole project is divided into multiple sprints each having duration of 4 weeks maximum. The teams are then assigned to work on each sprint.
- **SCRUM Meetings:** Daily meetings are held at the same spot every day and at the same time. The duration is about 15-20 minutes each meeting. The daily agenda is discussed as well as any problems.
- **Phases:** The SCRUM development methodology consists of the five main phases below.
    - Review release plans.
    - Distribution, review, and adjustment of product standards.
    - Sprint.
    - Sprint review.
    - Closure.
- **Sprint:** The sprint is the development phase where the actual work is done by the team. It has no sequence. The team decides what to work on, when, and how to accomplish the tasks. There can be more than one sprint in a project.
- **Sprint Review:** Each Sprint is reviewed upon completion. The completed sprint is reviewed and changes that are identified are added to subsequent sprints. The reviewing team consists of management, project stakeholders, sales and marketing heads, and at times customers.

- **Closure:** At this time the project is completed and ready to be deployed. Debugging is done and final adjustments are made to the product before deployment. Customer feedback is gathered and the teams that were formed are disbanded for other projects. Because of the unpredictability of changes it is not possible to determine the exact closure time of the project.

SCRUM Artifacts

SCRUM consists of four basic artifacts. They are listed below (Sutherland, 2010).

- Product Backlog.
- Release Burndown.
- Sprint Backlog.
- Sprint Burndown.

Product Backlog and Release Burndown

The product backlog contains the requirements of the project for the team to work on. The product owner is responsible for the development of the product backlog and providing all the necessary details needed to start the project. Additionally, the product owner also prioritizes tasks thereby making it easier for the team to know what tasks to work on first. The product backlog initially contains only the basic requirements. It is modified over the course of the project to incorporate changes made by the client and it exists as long as the project exists and works as a guideline for the delivery team. The requirements in the product backlog are usually stated as user stories making it easier to understand and comprehend. It contains everything from basic requirements to bug fixes. Additionally, the product backlog items have attributes like priority, description, and estimate. Priority is assessed on the basis of risk, value, and necessity (Sutherland, 2010).

The highest priority is given to the most important task that cannot be delayed. The team then begins development immediately on this task. Higher priority also, generally, indicates clear requirements and estimations. These tasks should have more detailed documentation than tasks with lower priority. Additionally, activities having high priority may require more than one sprint to be completed (Sutherland, 2010).

As mentioned, requirements most often are constantly changing. A product backlog records these changes and presents a new product backlog to be used as a guideline. Factors like market changes, economic conditions, consumer preferences, and labor cause changes in the product backlog. It is a living document that grows and expands with the growth of the project. Acceptance testing is often a cause of changes as well and is included in the product backlog. Acceptance testing is executed to determine if the customer is satisfied with the output or not. If the customer's response is negative, changes are made to the product as well as to the product backlog to satisfy the customer (Sutherland, 2010).

Multiple teams work on the project at the same time. Each team is assigned a different product backlog depending on the nature of the work they perform. For example, a team consisting of a technician and an engineer will perform technical work where as a team consisting or marketing heads will have different responsibilities to perform (Sutherland, 2010).

The Release Burndown, on the other hand, is a graph that records the remaining product backlogs and their estimated effort until completion. The estimated effort is in terms of the unit decided by the organization and the SCRUM team. Usually the unit is the time each sprint takes until completion (Sutherland, 2010).

Estimates of the product backlog are created initially but they often change later of course. The team is responsible for them along with the product owner. The product owner influences the estimates by providing the details but the final decision is made by the delivery team. The updated product backlog and release burndown are provided to the product owner to keep him informed of the latest activities on the project. Additionally, any changes are communicated to the product owner immediately (Sutherland, 2010).

Sprint Backlog and Sprint Burndown

The Sprint Backlog consists of the activities that must be completed by the team. It is developed after the product backlog. The Sprint backlog and product backlog should be in alignment with each other. It includes the work identified by the team as important to meet the sprint goal. Sprint backlogs are formed in the sprint planning meeting with the consent of each team member. Additionally, sprint backlogs should be decomposed appropriately to allow changes to be made to them if necessary (Sutherland, 2010).

Sprint backlogs are modified by the team during the sprints and the final product emerges at the end of the sprint. Once the work on a sprint begins the team may find out that more or less tasks are required and then the sprint backlog is adjusted accordingly to accommodate all the changes. Tasks that are found to be unnecessary are removed and the tasks of greater significance are given more attention by the team. A sprint backlog can be described as a real time picture of what the team intends to accomplish during the sprint. It belongs solely to the team and only the team has the authority to modify it (Sutherland, 2010).

Similar to a release burndown, the sprint burndown is a graph containing the amount of work left in the sprint backlog. The graph is created by estimating the amount of work left to be completed at the end of each day. It is owned and managed by the team and used to report to project stakeholders. This graph is an important tool for the team to report their progress and plan their work (Sutherland, 2010).

The goal of each sprint is of course to complete the required tasks so that the team can proceed on to the next sprint. There should be a functioning incremental software release ready for the client at the end of each sprint. Each incremental release should be tested completely and should support the previous and future product releases. An incremental release is marked as “Done” once it goes through all the development stages and passes acceptance testing at the client’s end (Sutherland, 2010).

Advantages of SCRUM Methodology

Every project development methodology has their respective advantages and disadvantages. Some of the advantages of SCRUM are listed below.

- Incremental and iterative method.
- Easy adaption for product and software development.
- Supports enhanced and easy communication.
- Participatory method.
- Maximum cooperation.
- Increased productivity.

Disadvantages of SCRUM Methodology

Some of the disadvantages of SCRUM are listed below.

- Small team. The work may be very slow because the team consists of only 7-8 members.
- Multiple requests. Different channels that requests are generated through might create confusion among the team.
- Development quality. With an increase in the number of team members, controlling the quality can pose a threat to the development process.

Chapter 8

Agile: Crystal

The Crystal project management approach is an Agile development methodology supporting iterative increments, continuous improvement, and close communication. The Crystal methodology is comprised of more than one development technique depending on the nature of the project. There is, however, support available that is used by organizations to create a Crystal methodology meeting their requirements (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Crystal is focused on the size, dimension, and attributes of a project. Each project is evaluated on these three aspects before deciding why the Crystal methodology should be applied. A more detailed Crystal evaluation matrix is below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

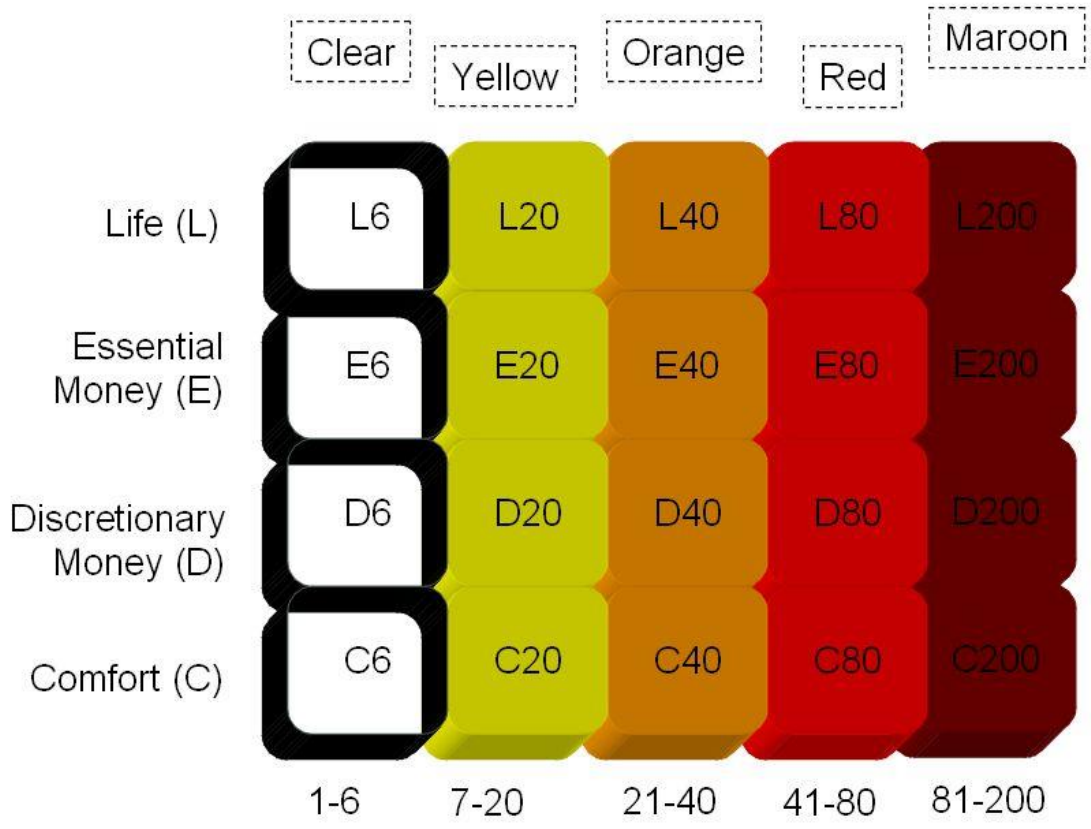


Figure 8.1 Crystal Family of Methodologies

Every project is different, therefore, before deciding on a Crystal methodology to apply, a project is evaluated on the basis of life, essential money, discretionary money, and comfort. Life in this case is actually the project's potential effect on human life. The evaluation result is then looked upon in the graph illustrated in figure 8.1. The darker shades represent complex projects where as the lighter shades represent easy projects. Complex projects require more communication and coordination therefore they are marked by colors like orange, red and maroon, whereas simple projects are marked by clear and yellow (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

The Crystal methodology was not designed for highly complex projects, but it can be used for them if deemed necessary. It was designed with versatility for small and medium sized projects (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004). The Crystal methodology consists of the three major characteristics listed below (Cockburn, Crystal methodologies, n.d.).

- **Human-Powered:** This means that the project's main focus is on the working team rather than on the technique applied. The project's success is achieved through the success of the team members involved. Therefore the team members are constantly encouraged to perform at their best and achieve the defined goals efficiently. Other methodologies are usually architectural-centric, process-centric, or tool-centric. However, the Crystal methodology is people-centric.

- **Ultra-Light:** This means that no matter what the requirements of a project are the Crystal methodology will always focus on reducing workload through effective management. Crystal strives to reduce the overhead required for production as well as documentation. This saves time and money.

- **Stretch-to-Fit:** This means that a project can be started with minimum requirements and more can be added later. Since the Crystal methodology is an Agile development methodology it supports incremental development making it easy to incorporate changes. It is considered safer and more convenient than freezing requirements all together.

The teams formed with the use of Crystal depend on the complexity of the project. For smaller projects team members consist of 6-8 individuals whereas for larger projects it can be as high as 50-100 members (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Crystal is actually a family of methodologies. Each methodology has its own name such as Crystal Clear, Crystal Orange, Crystal Red, etc. and they are all slightly different from each other. The selection of a Crystal methodology is made based on the criteria already mentioned above. The project outcome has to be safe and development efficient and the team members should be friendly and able to work with each other in harmony (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

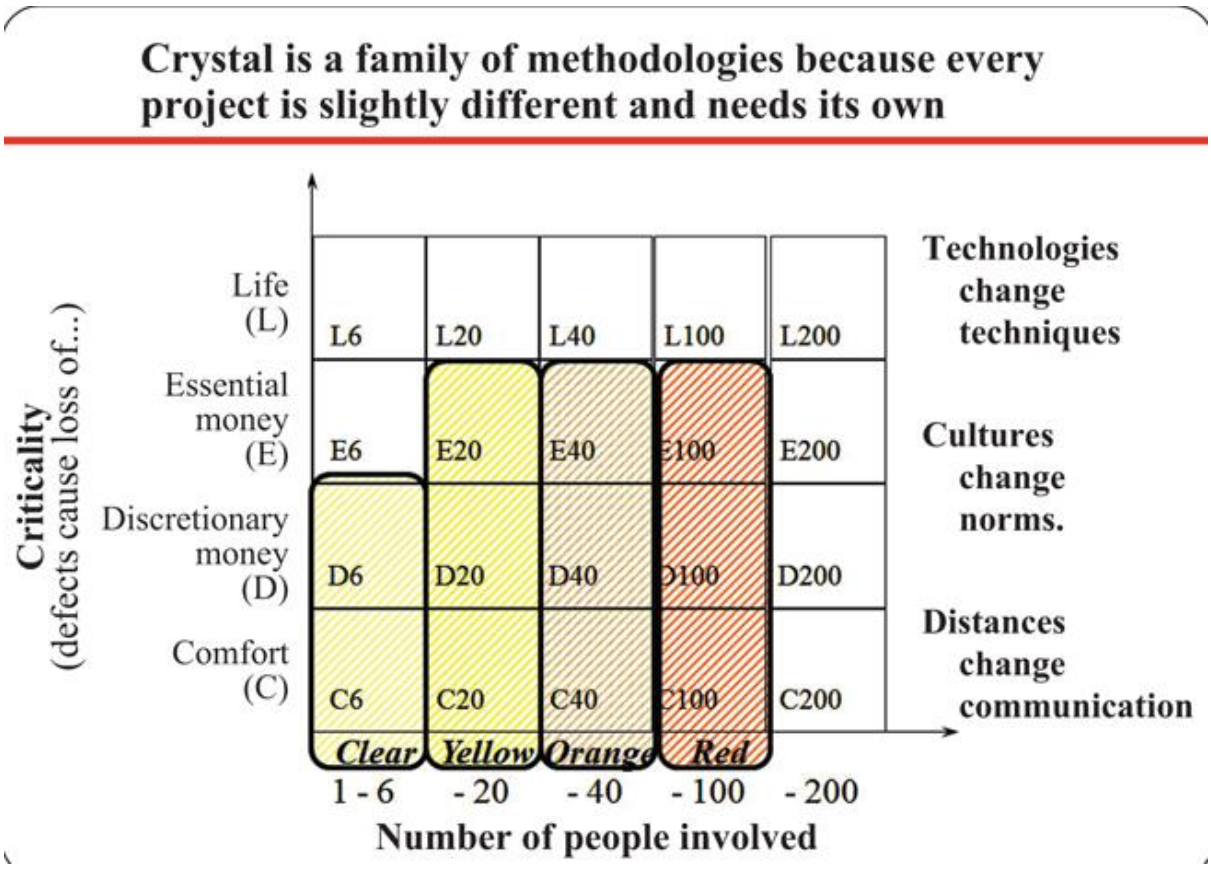


Figure 8.2 Crystal Family of Methodologies

Figure 8.2 shows another example of the Crystal family of methodologies. They consist of multiple strategies and techniques (Cockburn, Crystal Clear- A Human-Powered

Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004). Each of these will be discussed in detail later in this chapter.

- Exploratory 360.
- Early Victory.
- Walking Skeleton.
- Incremental Re-architecture.
- Information Radiator.

The family of Crystal methodologies is based on seven principles that will be discussed in detail later in this chapter. They are listed below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Frequent Delivery.
- Reflective Improvement.
- Osmotic Communication.
- Personal Safety.
- Focus.
- Easy Access to Expert Users.
- Technical Environment with Automated Tests, Configuration Management & Frequent Integration.

History of Crystal

Crystal was developed in 1991 when Alistair Cockburn conducted interviews with several project teams. He was asked to find an effective methodology that works best in all kinds of conditions. This resulted in a research study of multiple project teams and how they worked under different conditions. His research was based on multiple interviews with different team members and observation of their everyday tasks and activities (History, n.d.).

Through his study he revealed that no two projects are the same therefore they require a different team and work environment. He also stated that a "people-centric" methodology is better than a "process-centric" methodology because it should yield better results and be more efficient (History, n.d.).

After his long research effort he implemented these ideas in 1994 on a team consisting of 45 members. The cost incurred was about $15 million and the project had a fixed price and "Orange." He affirmed that his idea was indeed efficient and it yielded the desired output and also provided an abundance of creativity throughout the delivery of the project. With this work complete, he then defined the core success principles (History, n.d.).

In 1997 he published his research and "Orange" project management methodology under the title "Surviving Object Oriented Projects." Later in 1998 he formed a family of multiple methodologies and named it "Crystal" (History, n.d.). Crystal Clear was then published as a book in 2004 (History, n.d.).

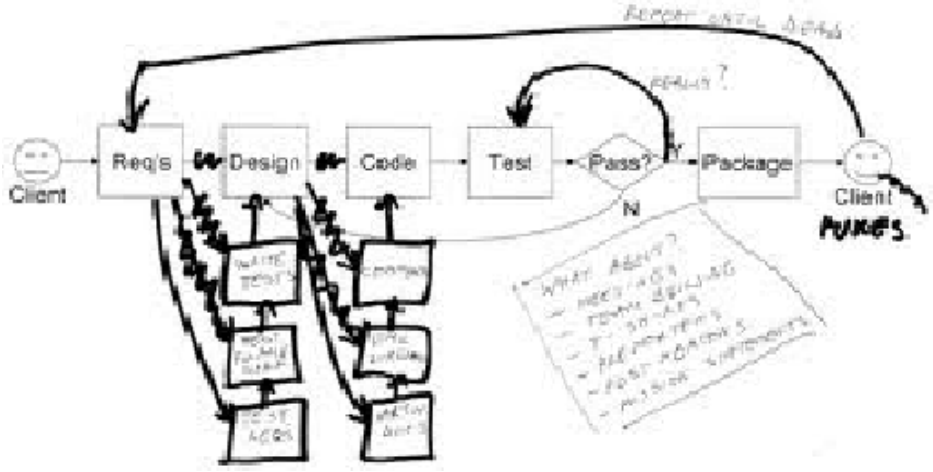


Figure 8.3 Rough Sketch Showing Crystal Implementation

Crystal Strategies and Techniques

The Strategies

1. **Exploratory 360:** Before beginning a project the team needs to evaluate the project based on certain criteria. If the project fails in any area it is either re-planned or an alternative approach is used for development. This process can take 7-10 days. The team needs to make sure that the project is both meaningful and is developed using the intended technology. The project is evaluated based on the criteria below (Cockburn, Crystal

Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

a. **Business Value:** The team and other respective stakeholders decide the purpose of the project and how it is going to benefit the organization and end user. Important user cases are developed, teams are assigned their responsibilities, and the final outcome is determined.

b. **Requirements:** Draft use cases are prepared and finalized with the consent of stakeholders and the end user before completing the final requirements document. The requirements document details of each requirement clearly and can be modified later if necessary. The Domain-Model is created from the draft use cases. It includes details about the purpose of the project, the technology used and the programming terms and vocabulary. It also includes the estimated size of the effort. The Technology used for development is prototyped and tests are run to see whether it will produce the required results. Experiments are conducted using the selected technology to ensure that time and resources will not be wasted. The Project Plan is then completed and reviewed by the Lead Designer, Executive Sponsor, and the Ambassador User to ensure it represents the business value and is suitable for use. In the end the final decision is reached either in the Reflection Workshop or Process Miniature.

2. **Early Victory:** Small victories throughout the project work as great motivation for the team. It gives them a sense of achievement and appreciation. This encourages them to work even more effectively and efficiently towards achieving the desired goal. This process should be developed early while forming the team and is the reason Crystal promotes the concept of early victories. During software development the first victory the team strives to achieve is a small piece of workable code. This deliverable is referred to as a 'Walking Skeleton' and consists of the minimum functionality. The deliverable may not account for much but it helps the team members get acquainted with each other's working style, assists with customer acceptance, and creates a sense of responsibility for the project (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004). Many problems the team might face are usually uncovered with accomplishment of this early victory. The initial task may be easy but later development may pose a lot of threats to the

success of the project. The team decides together how any problems will be dealt with when they arise. The strategy employed by the team usually is Worst Thing First. They deal with the worst possible problem first because afterwards everything else should be relatively easy. However, the problem with this strategy is that if the team fails to identify the actual problem at hand correctly, they could be creating solutions to a problem that doesn't actually exist. Because of this they may never be able to identify the true cause of failure. Additionally, the initial failures may dampen the spirits of the team hence resulting in lack of motivation and enthusiasm. Crystal supports the concept of Easiest Thing First, Hardest Second. It also encourages bringing people together who have never worked together before and giving them an opportunity to work with new people in a different environment. This generally helps each person to grow as an individual and subsequently flourish in their respective careers. The team is assigned the easiest task first and accomplishment generally gives them a sense of achievement creating motivation and determination for them. This also helps satisfy the project sponsors by assuring them that their resources are not being wasted. Once the team accomplishes the easy task, the difficulty level of each subsequent task is raised until the completion of the project. Any project that fails to deliver business value is a failure to the sponsors. Therefore it is of utmost importance to develop a strategy that strives to deliver a project that enchases the business value as well as meets all initial requirements. Using Crystal the team develops a strategy that ensures a project enhances business value. Even if the project is not completed on time but delivers business value the sponsors and stakeholders will be satisfied as their scarce resources will be put to good use and the final product will be expected to meet all requirements (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

3. **Walking Skeleton:** The Walking Skeleton is a fully functional miniature piece of code that provides end-to-end basic functionality. It is developed initially to test the requirements and the technology used. The client's feedback is solicited to make sure the technology used will eventually produce the required result. It also helps in finalizing the basic architecture and functionality. For every project a Walking Skeleton varies greatly depending on the kind of software being developed. For example, for a client-server system, the Walking Skeleton could be a simple screen-to-database-and-back functionality. This can also be described as a demo of what the actual software will

eventually look like. A Walking Skeleton is different from Spike. A Spike is the smallest kind of demonstration that doesn't contain any significant functionality and is discarded after approval. It is just used to determine whether the project is headed in the right direction or not. A Walking Skeleton, on the other hand, works as the foundation on which the whole project will be based. Each component is gradually added to the skeleton until the development of a complete package of code. However, while complete, there is still room for growth (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

4. **Incremental Re-architecture:** With changes in requirements the system architecture needs revision as well. It is very unlikely for a development team to halt the whole process just to incorporate changes, which is why the incremental re-architecture concept was adopted. The changes are incorporated with each iterative increment making sure the client is happy with the final result. The team revises the architecture of the software in stages while keeping the development process running smoothly. This is also used to evaluate the system's end functionality. The difficult question here to address is how the team should decide the extent to which the design of the infrastructure and architecture will be complete at the initial level. There is no definite answer to this question but a designer with experience in this particular domain can come up with the exact number of days depending on the complexity of the project. It could take up to 7 days or even more. The decision if also influenced by previous similar projects. Engineers using Crystal will not spend more than a week or two to reach a final decision. They start the work on a Walking Skeleton and the feedback from the client is used to improve the project architecture. Research has shown that creating a simple architecture initially and then gradually adding new features to it helps in building successful projects. Therefore, this concept is used in Crystal. Applying this strategy has its perks but may not be beneficial for all kinds of projects. Some of the possible benefits are listed below.

    a. A simple architecture is easy to modify if needed.

    b. The infrastructure and function teams work in parallel on improvements.

    c. The end user is able to view the increments and make any changes they feel necessary.

    d. Each increment is better than the previous enhancing business value.

e. The tests on incremental releases help minimize failure of the project.

f. Early releases can help a business generate revenue that can be used to facilitate the development of future releases.

5. **Information Radiators:** Information Radiators are posters hung around the team's working area. It is basically a large piece of paper hanging on the wall with requirements and daily activities on it. An effective information radiator has the characteristics below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

   a. It is large and easily visible to everyone.

   b. It is easy to comprehend at a glance.

   c. It is changed periodically to incorporate changes made by the client.

   d. It is easy to update regularly.

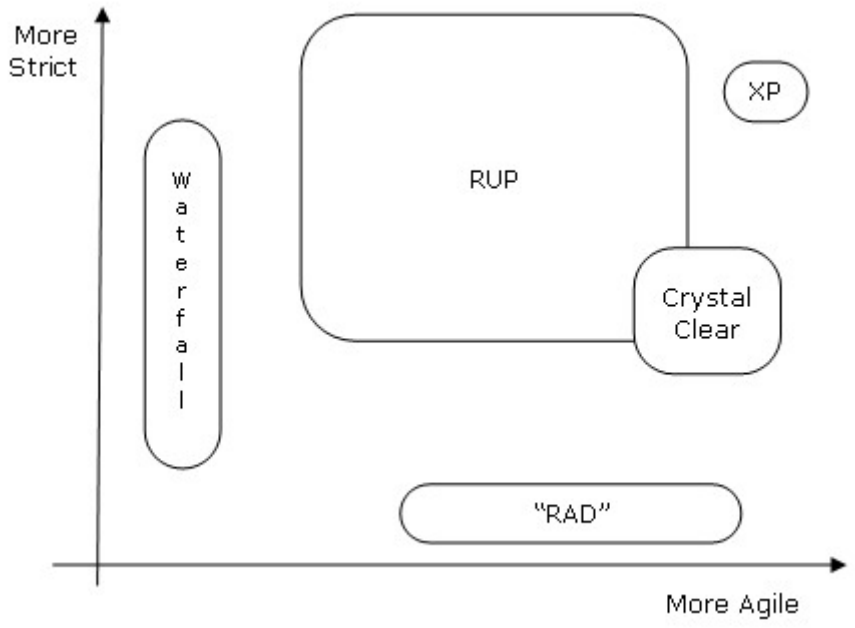


Figure 8.4 Figure Shows How Crystal is "More Agile" in Comparison

6. While usually it is just a piece of paper hanging on a wall of the team room, sometimes it is a web page that is updated and viewed by the team regularly. The purpose of the information radiator is not only to communicate the requirements to the team but also to

other people in the organization. However, not all details are written and posted on the wall. Simply, the basic scope and purpose of the project is documented to create awareness among the people in the organization. Since the team has a close communication network, they typically do not need information radiators. That said, they can be used for all kinds and sizes of projects making communication effective and efficient. History has shown that generally speaking small teams use information radiators more effectively because it is easier than updating a website. As mentioned, information radiators usually show the status of a project, but also contain the additional information listed below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

a. Current work on the incremental release (user cases and user stories).

b. Current work assignments.

c. Number of tests written or passed for the incremental release.

d. Domain model of the incremental release.

e. The status of the incremental release.

Often, online portals and web pages do not make effective information radiators because they require effort on the part of the viewer. Usually large poster boards are placed around the team room and additional graphs, charts, use cases, and user stories for the project are also placed in plain sight. This is very effective for operations because any team member needing information can view it instantly.

The Techniques

Crystal really doesn't have any specific predefined techniques used for development purposes. Instead, it uses a combination of multiple techniques to get effective results. There are 9 techniques in all that were selected by Alistair Cockburn and they are listed below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Methodology Shaping.
- Reflection Workshop.

- Blitz Planning.
- Delphi Estimation.
- Daily Stand-ups.
- Agile Interaction Design.
- Process Miniature.
- Side-By-Side Programming.
- Burndown Charts.

Methodology Shaping

Methodology Shaping is made up of two important factors discussed below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

1. **Project Interviews:** People with past project history are interviewed to learn from their experiences and how it can be used in their current project. The project team interviews other members of the team that are currently on or have previously worked on different projects within the organization. This helps to get insight on their work experience and work environment. Each member of the team should at least interview 5-10 other members. The main people to interview are the Project Manager, the Team Lead, the Programmer, and the Interface Designer. They will all probably have different perspectives for the same project that will prove to be very useful. During an interview, there are simple steps one can follow to avoid deviation from the topic and they are listed below.

    a. **Step 1:** Ask for work product samples to evaluate them. Determine how long each work product will take and what the final output will be like. Ask about the techniques that will be used while producing work products and how documentation will be maintained and updated. Also ask whether the communication between the team members will be formal or informal. All of this information will be very useful in helping to plan and manage the project.

b. **Step 2:** Ask for a short history of the project they are currently working on or one they worked on in the past. The history of the project should include data structure, team size, turnover rate, and emotional high's and low's.
c. **Step 3:** Ask about any and all mistakes made during the delivery of the project and what can be done better with the current project.
d. **Step 4:** Ask about everything that was done right on their last project and how those activities can be incorporated into the current project. This should include the actual work on the project, the friendships formed, the good memories, the software design and more.
e. **Step 5:** Ask how their past project was prioritized and what was as the basis for the prioritization. Also ask if anything about the project was surprising or unexpected.
f. **Step 6:** Ask them to generally comment on their past or current projects. This question usually receives all kinds of different responses hence providing more information.

It is helpful to construct an interview template before conducting the interviews and make changes to it, if necessary, as the interviews progress. Below is a template that can be used.

1. Project name, job of person interviewed
2. Project data (start / end dates, staff size, domain, technology).
3. Project history
4. Did wrong / would not repeat
5. Did right / would preserve
6. Priorities
7. Other

Figure 8.5 Interview Template

2. **Methodology Shaping Workshop:** This is conducted with a group of people who are not aware of their respective responsibilities and operational duties. The output is a list of rules and policies that will be used to deliver the project and manage the tasks to be performed. Any ideas generated by the members are thoroughly discussed and a consensus is reached whether to accept or reject the idea. The list of ideas and rules is generally long at this point. It is reviewed to capture the relevant ones and ignore the rest and the list is updated as the project progresses

(Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Reflection Workshop

Reflection Workshops are conducted to review the work and to discuss areas that need improvement. These workshops are conducted at regular intervals and are very beneficial, especially during the first few months of the project. The team often has a lot to discuss, but gradually these workshops will become shorter and shorter in duration as the project progresses and team relationships mature. The three major areas discussed in a reflection workshop are listed below.

- What to keep and what to ignore.
- Any problem areas are discussed.
- What to try in the next incremental release.

Blitz Planning

The Blitz planning session is an opportunity for executives and stakeholders to collectively discuss the road map for the project. Blitz planning is also called a Project Planning Jam Session and each individual member of the team participates in making improvements to the project. All the executives and stakeholders are gathered in a room and every task that needs to be completed is written down on index cards. They are then arranged in a sequence, prioritized and dependencies are determined. Having every task on the table at the same time helps determine the milestones and deliverables for the project. The steps below are part of the Blitz planning process in Crystal (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Gather the Attendees.
- Brainstorm the Tasks.
- Layout the Tasks.
- Review the Tasks.

- Estimate and Tag the Tasks.
- Sort the Tasks.
- Construct Walking Skeleton - Earliest Release and Quickest Revenue.
- Identify Additional Releases.
- Optimize the Plan to fit Project Priorities.
- Capture the Output.

Delphi Estimation using Expertise Rankings

The most accurate estimation method for a project is from the bottom up. Each activity is individually estimated before calculating the total estimated time for the project. The steps below can be used for this purpose (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Estimate the size of the solution to be delivered.
- Estimate working time according to the person working on the project.
- Determine releases according to their technical and business dependency.
- Determine the gap between each release.

Daily Stand-Up Meetings

A daily regular meeting is conducted not to discuss problems but to identify solutions to the problems. Not every member has to attend the daily stand-up meeting. Those having problems attend the meeting and discuss their issues to find a solution. The status and progress of the project is also discussed. This meeting focuses on the three critical areas listed below (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Yesterday's Work.
- Tomorrow's Work.

- The Problems Faced.

The Essentials of Interaction Design

Interaction design is important because it keeps the project driving in the right direction. The process explains each and every task and how the tasks are interdependent and link with each other. The areas listed below are key to this practice (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

- Essential Interaction Design (the Workshop).
- Deriving the UI.
- Usability Inspection (during Design).
- QA Testing the System Personalities.

Process Miniature

Complex projects, oddly enough, take longer for every team member to fully understand all the requirements and perform efficiently. The smallest of mistakes can lead to major failures. This is why Crystal supports the idea of developing a miniature process model for the overall project to evaluate potential problems that could occur (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Side-By-Side Programming

This technique is very simply paired programming (i.e. two people work on the same piece of code side-by-side). This speeds up the overall process and avoids stress caused by work overload with just one programmer (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Burndown Charts

Burndown charts are made up of the user stories and they show the amount of work that is being completed and how quickly it is being finished. They are great reporting tools and promote discussion of the scope of the project. Additionally, they are one of the most

common and effective tools used in project management. Finally, they provide a simple means of constant guidance and communication of the delivery effort (Cockburn, Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects, 1998-2004).

Advantages of Crystal

There are number of advantages to using Crystal and they are listed below.

- Crystal has been proven to saving time and money.
- Since Crystal is an Agile life cycle methodology, it is generally easier to correct mistakes on a project.
- Constant feedback with Crystal makes dealing with changes easier.
- Crystal has been shown to decrease project time to market.
- Crystal projects can typically reduce project overhead costs.

Disadvantages of Crystal Mythology

Along with advantages of using Crystal, there are also disadvantages that are listed below.

- Crystal does not place emphasis on documentation.
- Estimates for large projects can be miscalculated in terms of effort, time and resources needed.

Chapter 9

Agile: Dynamic System Development Method (DSDM)

The Dynamic System Development Method is yet another Agile project delivery life cycle. DSDM was initially used just for software development but is now being used for process engineering, business development, and general problem solving projects.

Brief Description

DSDM is ultimately and extension of the Rapid Application Development (RAD) method and was first released in the mid-1990s. DSDM was initially adopted in England and then became popular in Europe before finally emerging in the United States. Since its release, DSDM has proven to be an effective approach for many companies.

Like all Agile life cycle processes, DSDM has also evolved over the years. According to Dane Falker, while the abbreviation remains the same the name has been altered. He states, the first "D" in DSDM still stands for "dynamic" but the "S" stands for "solutions." "Solutions" in this case is generally a reference to business solutions. He goes on to say, that the second "D" reflects "delivery," which has a broader meaning than "development" and last the letter "M" can be most appropriately represented by the word "Model" rather than "Method." These new words for the DSDM acronym better reflect the importance of the "Dynamic Solutions Delivery Model" from Dane's view point. The nine principles of DSDM include active user involvement, team decision making, frequent delivery, and testing throughout the life of the project. Thus arises the idea of an empirical development approach (Highsmith).

What is DSDM?

Ultimately, the Dynamic System Development Method is an Agile project management life cycle that specializes in delivering the right decision at the right time. For many years DSDM has been the leading Agile approach that provides rigorous process with agility and flexibility as demanded by various organizations. The emphasis is on defining and focusing on strategic goals and early delivery of results for the business.

History of DSDM

DSDM first appeared in 1994 through a joint collaboration of professionals in software engineering and suppliers. At that time it was named DSDM Consortium (Dubleowseven, 2014). April 2007 saw a rebranding of this model under the name of DSDM Atern. The name was a shortened form of Arctic tern that refers to a collaborative bird (DSDM, 2012).

Since then it has widely been used for project management around the world. Its focus is primarily on solving people related problems in project solution delivery. DSDM was based on Agile principles and later amendments surfaced in April 2008. Finally, in 2014 Atern branding was removed and DSDM Agile Project Framework is now just referred to as DSDM (Dynamic systems development method, 2016).

The DSDM Process

The DSDM project life cycle has five phases that are listed below.

- Feasibility Study.
- Business Study.
- Functional Model Iteration.
- System Design and Build.
- Implementation.

Figure 9.1 represents the DSDM life cycle process that is also referred to as “the three pizzas and a cheese.” Each of the major phases is an iterative process. The use of three iterative models for DSDM may initially seem ambiguous, but once understood, they can be used to make flexible project plans.

Feasibility Study

This phase evaluates whether or not the DSDM life cycle is feasible or right for the project at hand and is different from a traditional approach to a feasibility study. Generally a waterfall approach is considered as the default for software projects so when DSDM is suggested as an alternative by internal or external IT suppliers then it becomes essential to make sure that it is indeed appropriate.

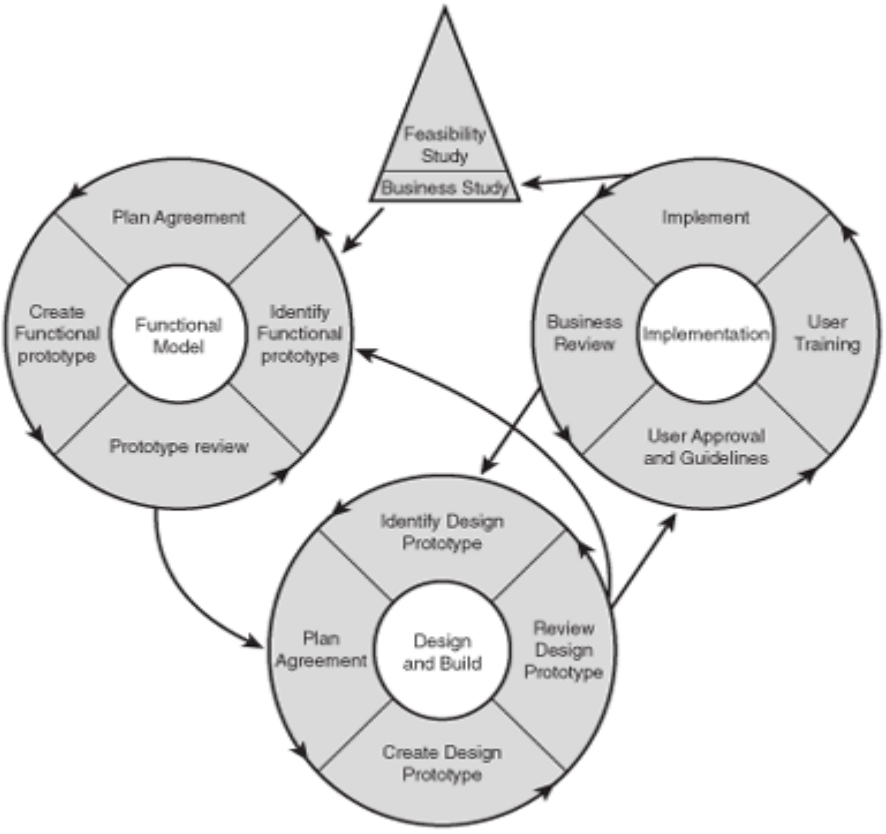


Figure 9.1 DSDM Life Cycle

In the feasibility study phase, the usual factors are also reviewed. These items include answers to questions like "Is the proposed system possible technically?," "Can current business processes accept its impact on them?," or "Is DSDM the best way to build the system?." Often the factors that drive these decisions are organizational and people issues. (Stapleton, 1997)

History has shown that DSDM is best used for the development of systems that are needed urgently and in this case the feasibility study is essentially a short exercise to complete. Some organizations, however, are more cautious about these studies than others. In their case a feasibility study can take a long time before its final acceptance by all parties. Frequently, many people can suffer with analysis paralysis while trying to ensure that all the factors of a problem have been reviewed. The DSDM feasibility study will typically only last for a few weeks. Additionally, the feasibility report will consist of all the usual topics, but not in a descriptive manner. Quite simply it is meant to detail whether the project is technically viable or not.

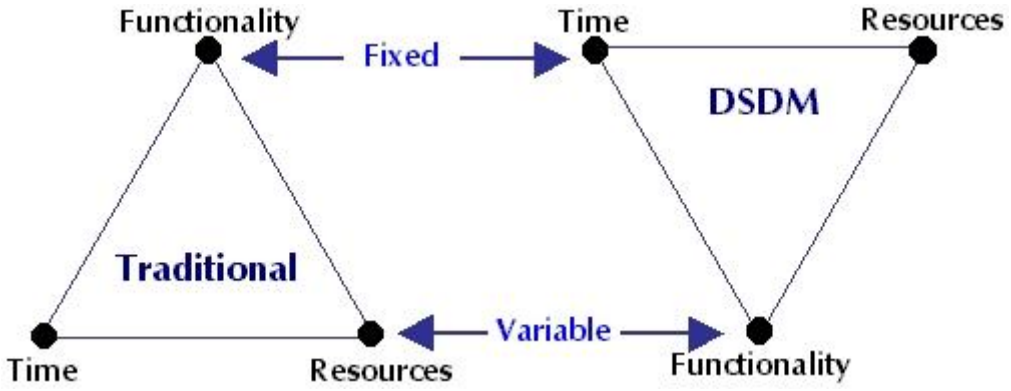


Figure 9.2 Figure Shows DSDM Approach vs. Traditional Approach

The Business Study

After deciding that DSDM is indeed the path to take, the business study is started to define the basis for all project work. This too is not a huge effort and is meant to gain enough information on the business and associated technical constraints to proceed safely. Judging only from its name, one could think this is a major activity. However, the focus is merely to gain an understanding of the business' processes and information needs. In order to complete this task in the shortest possible time collaboration is needed. Interviewing people would simply not work because of the time required. To collect this information DSDM uses a series of workshops with the company's experts. As a result of the workshops, a Business Area Definition is generated that identifies the business' processes and its information needs.

A Business Area Definition presents a high level view of the business processes to be automated. In a structured environment, it will contain data flow diagrams and in an object-oriented environment it would contain a business object model.

Since code will start to be produced in the subsequent phase, it is imperative to know not only the functionality of the end product, but also the architecture of the system to be used. Therefore, another product that results from the business study is a System Architecture Definition. This effectively defines the architecture of the software with a description of the development and target platforms. Much like everything else that is produced with the completion of the business study, the System Architecture Definition can be changed at a later time.

Last but not least, the outlined plan generated as a result of the feasibility study is refined in order to produce an Outline Prototyping Plan. This refined plan details all activities involved in subsequent phases.

Functional Model Iteration

The Functional Model Iteration phase is one of two phases of the DSDM life cycle that are iterative. The major focus in this phase is building the prototype with an iterative approach and getting feedback from users in order to meet the requirements. By involving the users through demonstrations of the prototype and receiving feedback from them, the prototype is gradually improved. The whole feedback and modification process can be repeated until the elements of the functional model are agreed upon. The ultimate product of this phase is a completed functional prototype that contains all the elements consistent with the major functionality.

Design and Build Iteration

In this phase, the focus is on ensuring that the prototype produced is adequately engineered to meet the operational environment. Each component of the prototype is further refined during this phase until they reach the accepted requirements. The final deliverable of this phase is a fully tested product of that is ready for the Implementation phase.

Implementation

The final phase in the DSDM life cycle is the Implementation phase. The users receive training in this phase and the results of the project are operational in the intended environment. The four possibilities for this phase are listed below.

- Everything is delivered according to the requirements.
- A new functional area may be uncovered, so there may be a need to repeat the whole process.
- Areas of lowest importance may be released later because of time constraints for the project and the project will return to the functional model iteration phase.
- It may also be possible that some non-functional requirements were not fulfilled and causes the project to go back to the design and build phase.

Due to the incremental approach of DSDM, it is expected that all development stages may be revisited. That being said, all phases are only completed to the point that the project can move to the subsequent phase because work can be completed in a later incremental release.

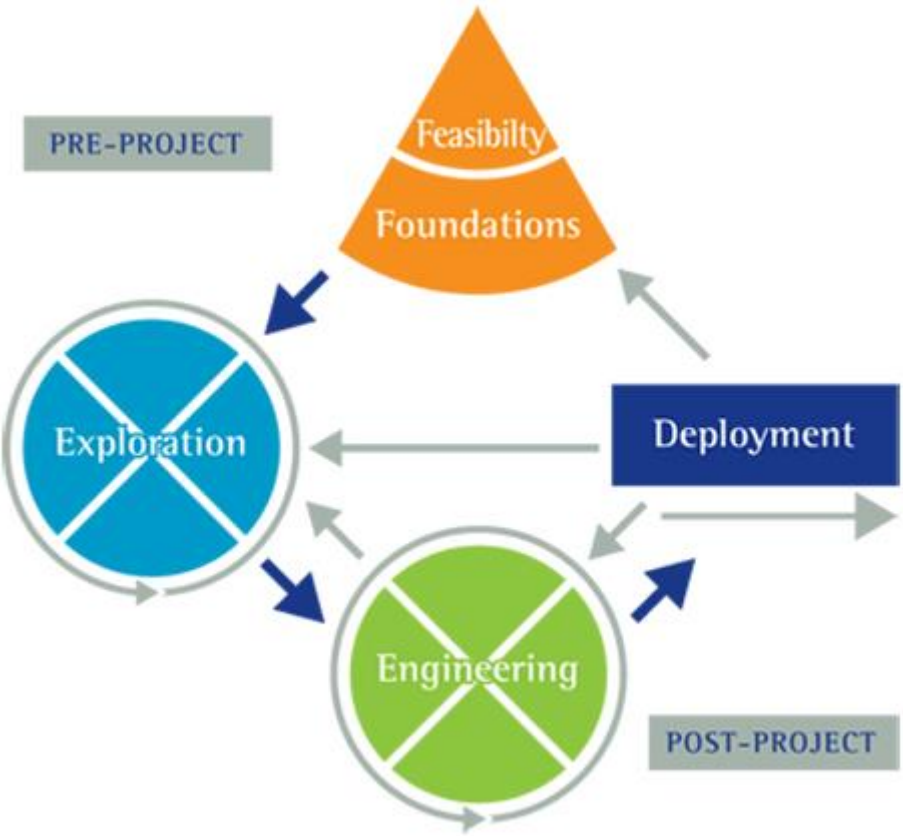


Figure 9.3 Additional Graph of DSDM Life Cycle

Principles & Values

The framework of DSDM can be used for both Agile and traditional methodologies. However, in order to truly understand the relationship of DSDM to the Agile methodology it is necessary to review the core Principles of DSDM. These principles are discussed in detail below.

The Nine Principles

For any DSDM implementation, there are nine essential principles. These principles are important enough that if one of them is missing it can create complete failure in the framework philosophy and increase project risk significantly.

- **Active User Involvement is Imperative:** This principal is regarded as the most important one of the 9. This is because history has shown that user engagement throughout the project tends to reduce errors through user observation thereby reducing cost. The project rules of DSDM recommend working with small groups of users versus large groups for more effective engagement. The selection of these users can be made on a continual basis rather than in one review session. The attribute of Continuity is also demanded by DSDM principles.

- **Teams Must be Empowered to Make Decisions:** For a project to move as swiftly as possible communication problems between contributors and managers must be avoided. The practice of demanding authorization of every decision slows down projects considerably. To address this issue users and project participants are empowered to make decision related to the areas below.

    - Requirements.
    - Content of Incremental Release.
    - Requirement and Feature Prioritization.
    - Details of Technical Solutions.

- **Focus on Frequent Delivery:** Incremental deliveries facilitate rapid error detection allowing for errors to be corrected earlier in the project. This is true for both documents and code, which includes items such as data models and requirements.

- **Fitness for Business is Criterion for Accepted Deliverables:** The DSDM framework shows that its prime purpose is the delivery of software that can solve business needs and can be enhanced in later incremental releases.

  Part of a software system's natural life cycle is refactoring, and feature enhancement and these should be considered a vital part of the project rather than just a task. DSDM focuses on satisfaction of business needs first and does not support the writing of ad-hoc code. Only after satisfying business needs does focus shift to refactoring and associated activities. This is true even if it is difficult to identify business needs due to the complexity of the project.

- **Iterative and Incremental Development is Mandatory:** In DSDM, like other Agile methodologies, the project deliverable is decomposed into incremental feature packages to manage project complexity. Until all the business requirements are satisfied each incremental release adds new features. Using this principle, it must be accepted that any code is subject to change. Even in the beginning of the project this principle can be executed and specifications and other work products can be incrementally completed as well. The smaller the incremental release, the easier it is to make changes.

- **All Changes during Development must be Reversible:** To achieve responsiveness changes must be able to be completed during any increments' development. The project's dynamic configuration is supported by advance software tools. It is often feared that if any development is reversed it will result in loss of previous precious work, but DSDM supports development of smaller incremental releases to limit the loss of work.

- **Requirements are Base Lined at High-Level:** While free to alter requirements during the project, some high level requirements are needed. This baseline requirements list is understood to be a requirement "freeze" and is decided upon during the business study phase of the project.

- **Testing is Integrated Throughout the Life Cycle:** In many project life cycles testing is performed in the final stages. In DSDM projects, testing is required in the early stages of the development process. This could just mean peer reviews and cross checks of test plans, interviews of a control group, or other similar techniques.

- **Collaborative and Co-operative Approach:** Encouraging teamwork within the project is a significant aspect of DSDM. It is mandatory for staff members to be present, because cooperation is critically important to the success of a DSDM project. DSDM is based on collaboration to attain success and many difficult tasks would be even more complex if the atmosphere was not trustworthy or honest. An honest environment results in the collection of improved feedback resulting in project efficiencies. (Martin Wieczorek, 2001)

Philosophy of DSDM

The framework of DSDM is a modular and dynamic system in itself. During development, DSDM designers were concerned with the "edge case" rather than a simple project "ingredient", meaning they felt it was more important to know what changes to make and still be in conformance with the project's time constraints.

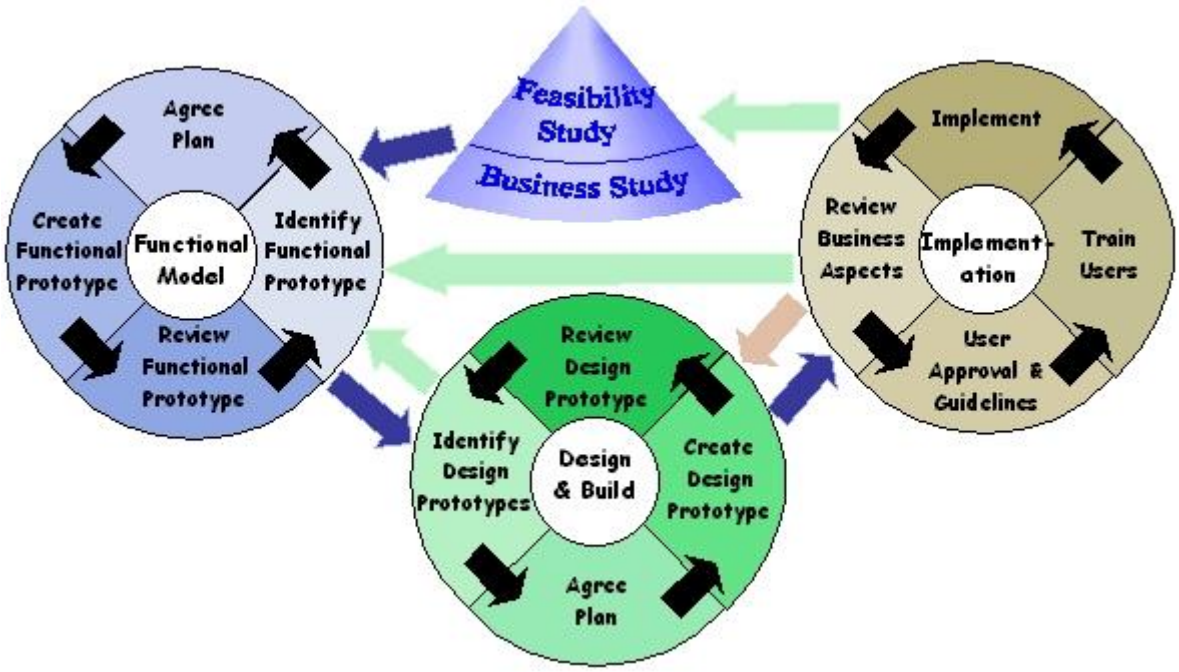


Figure 9.4 DSDM Implementation

While it is not required for a whole project to be implemented using DSDM, what it is required is a strict obedience to the above mentioned nine principles. Apart from that, the Agile development processes can be implemented by any project manager, depending on the situation and constraints, and DSDM can even be combine with other methodologies is so desired. The core idea of DSDM is that frankly, "On the first day of development no plan will ever survive" and this is why it supports many techniques to handle these dynamics. To sum it up, DSDM was not created to solve all development problems, and DSDM is not the best life cycle to choose if the 9 principles are not implemented.

Is DSDM Still A Good Choice to Make?

DSDM principles are ultimately built on a number of current best practices that where gathered, synchronized, and rolled up into a single methodology. This was exactly the focus in 1994 when the first release was made. While version 4.2 of DSDM is still in use and

supported, the current version is DSDM Atern. Atern stands for Artic Tern, which is a bird that inspired the name. DSDM will always remain a life cycle built on best practices, but many organizations adapt the DSDM processes to their environments, requirements, and projects.

Roles and Responsibilities

A DSDM project consists of 7 stages that are managed and embedded with an augmented set of roles and responsibilities that are supported by several key techniques. Additionally, the foundation of any successful project is an effective team. With this in mind, DSDM assigns each person associated with the project roles and responsibilities. Below we discuss those roles.

Business Sponsor

This is most senior business role on the project. The Project Champion is the Business Sponsor who is committed to the project, delivery approach, and the planned solution. He is responsible for the Business Case and more specifically once the solution is delivered he will own it and be responsible for understanding all of the benefits associated with it. The Business Sponsor must be someone high enough in the organization that he can resolve business problems and make financial decisions. A key responsibility of this role is to ensure and enable swift progress of the whole project. Only one person should be placed in this role and they must be available for the entire project thereby providing a clear escalation path.

Responsibilities

- Owns the business case for the project.
- Ensures viability of the project with reference to the business case.
- Ensures funding and other resources are available for the project.
- Ensures fast, effective decisions for all escalated project problems.
- Generates quick responses to all escalated issues.

Business Visionary

This is also a senior business role for the project. This role is involved in the project more actively than the Business Sponsor. The Business Sponsor’s needs are interpreted by

the Business Visionary, and then communicated to the team. Additionally, where appropriate, the Business Visionary ensures the Business Sponsor's needs are represented in the Business Case. The project team takes direction from the Business Visionary as he ensures the benefits that are detailed in the Business Case are achieved.

Responsibilities

- Owns, from an organizational perspective, the wider implications of any business changes.
- Defines vision for the project.
- Promotes and communicates the defined business vision.
- Monitor's project's progress with respect to the vision.
- Contribute to review sessions, design, and requirements.
- Approves changes to high-level requirements.
- Ensures the availability of business resources as needed.
- Drives collaboration across stakeholders.
- Promotes transition of the vision into the work environment.
- Arbitrates team disagreements.

Project Manager

The Project Manager is responsible for the delivery of the solution. A critical function of this role is managing the work environment for the solution. At a high level, the Project Manager coordinates all management aspects and he works in conjunction with the team leaders for more detailed planning of the solution delivery. Even though the Project Manager is focused on delivery, this does not suggest from what department in an organization they should come from. The Project Manager is an important role and he must be available during the whole project.

Responsibilities

- Communicating with the project governance team and senior management.

- High-level project planning and scheduling.
- Managing project progress against baselines.
- Monitor issues and risks and escalate as necessary.
- Motivating the teams.
- Resourcing specialist roles.
- Coaching the Solution Development Teams on difficult situations.

Technical Coordinator

This role ensures the Solution Development Team works in a consistent manner and that the project meets the required technical quality standards. Additionally, this role provides advice on technical decisions and innovation and in doing so helps hold the project together. Ultimately, this is the same role from a technical perspective as the Business Visionary's role is from a business perspective.

Responsibilities

- Controlling and agreeing the architecture.
- Identifying the technical environment.
- Ensuring that non-functional requirements are met.
- Managing each team's technical activities.
- Controlling the configuration of the delivery.
- From a technical perspective, manages the transition of solution into the business environment.
- Managing technical differences within the teams.

Team Leader

A Team Leader reports to the Project Manager and manages the delivery of objectives for a particular Solution Development team. Often, Team Leaders only lead during a particular phase and have an additional team role during the project. The Team Leader

coordinates the team and does the detailed planning of the product's delivery. It is possible that the Team Leader can even be different from one time-box to another depending on focus.

Responsibilities

- Ensures an on-time delivery of the team's objectives.
- Encourages and motivates the team members.
- Manages the project life cycle process within the team.
- Drives testing and review activities within the team.
- Manages the issues and risks within the team and escalates as necessary.
- Monitors daily progress of team.
- Reports team's progress to the Project Manager.
- Holds daily team meetings.

Business Ambassador

The Business Ambassador is the business role associated with the business area that will implement the solution. This role guides the delivery of the solution and communicates other's input and ideas as required. A Business Ambassador must have the authority and knowledge to ensure the end solution meets the business needs. While he may not be a senior manager within the company, he must be empowered and able to participate on the project as required.

Responsibilities

- Contributes to all review sessions, design and requirements.
- Provides daily business perspective.
- Provides the business scenarios for defining and testing the solution.
- Communicating with stakeholders as needed.
- Communicating daily assurance the solution is progressing per plan.

- Responsible for user documentation.
- Manages user training.
- Daily meetings.

Business Analyst

This role is significant to begin with, but has increased importance in IT projects. The Business Analyst acts as a bridge between the business people and the technical team. On a daily basis, he ensures that the Solution Development Team is proceeding down a successful path and is of one mind on the project. Business needs have to be completely understood before they are implemented and this is the responsibility of the Business Analyst. While this person is not an intermediary between the Developers, Business Ambassadors, Advisors and others, he does help facilitate communication among the team.

Responsibilities

- Acts as liaison between the business and technical people.
- Ensures proper communication within the Solution Development Team.
- Ensures the business requirements are completely understood and implemented.
- Guarantees that daily business decisions are efficient and effective.

Solution Developer

Business requirements are interpreted by the Solution Developer. He translates requirements into a solution that is deployable and also meets non-functional and functional needs. Ideally, the Solution Developer is allocated to the project full time. If this is not possible, the project needs to be his first priority or there will be significant risk to the project. These risks need to be managed by the Project Manager.

Responsibilities

Works with Solution Testers and Business Management on the activities below.

- Deployable Solution.
- Models.

- Record changes to requirements.
- Changes made to any interpretation of the requirements that result in re-work.
- Adhering to best practices in the Technical Implementation Standards.
- Participating in quality assurance efforts to ensure that products truly meet the requirements.
- Testing their work prior to independent testing.

Solution Tester

This role is completely integrated with the Solution Development Team and during the course of the project completes testing in accordance with the Technical Testing Strategy.

Responsibilities

- Consults on building test cases.
- Executes various tests.
- Creation of test products.
- Supporting the Business Advisor and the Business Ambassador for critical tests.

Business Advisor

A Business Advisor is often a peer of the Business Ambassador. He is normally an intended user of the solution and provides specific input for solution development and testing. However, sometimes his sole role is to provide legal or regulatory advice that the end product needs to comply with.

Responsibilities

- With regard to his specialty, provides input into reviews, design, and requirements.
- Inputs on daily project decisions.
- Provides expert input on acceptance testing.

- Provides input on user documentation for the solution.
- Provides input on user training.

Workshop Facilitator

This role of Workshop Facilitator is to manage the workshop process and drive communication. While responsible for the context of the workshop, he is not responsible for the content and ultimately he should be independent of the outcome of the workshop.

Responsibilities

- Planning the workshop.
- Agreeing on the scope of the workshop.
- Familiarization with the subject of the workshop to be held.
- Evaluate if the participant is suitable for the workshop.
- Ensures participants understand the objectives of the workshop.
- Facilitates the workshop.
- Reviews workshop execution.
- Drives completion of any workshop preparation.

Atern Coach

The Atern Coach is a process expert. As such he should be certified in the process and his job is to guide a team with limited experience using Atern. The Atern Coach ensures that processes are not just blindly followed and assists in problem resolution as it pertains to the DSDM life cycle. Essentially, there are two ways to resolve process problems. You can either look for ways for the process to be effective in the business environment or adapt and substitute another technique. Either way, the Atern Coach is the DSDM process expert that guides this effort.

Responsibilities

- Provides knowledge and experience to inexperienced teams.

- Tailors the DSDM life cycle for the project environment.
- Champions the DSDM life cycle.
- Supports cooperative teamwork demanded by all Agile approaches.
- Builds Atern capability.

Specialist Roles

Every project has its diverse needs and they need to be handled accordingly. The above mentioned roles may or may not cover every responsibility for a specific project. As such the Project Manager may find a need to hire someone as a Specialist for certain duties that cannot be handled by other members of the Solution Development Team. The tasks assigned to such person(s) often vary according to the scope and demands of the particular project. Additionally, there are other considerations for integrating any new Specialist into the team. In short, it is important to communicate their work responsibilities so that they can fulfill their role on the team and know what is expected of them. If the requirements for the Specialist are clearly defined, it is much easier to identify and acquire the right person for the job.

Team Organization and Size

Usually a DSDM project is built up of one or two teams, so that one team can focus on development of the product and the other team can focus on testing of the product. Organizational research shows that approximately 5 people for a single team are ideal. If the workload is more than a single team can handle, then a multi-team structure must be developed. Many projects using DSDM have up to a total of 150 people. For large multi-team projects decomposition of tasks must be completed with team functions in mind. Historically, many projects have had success by using a testing team and a development team for each single feature set. Since testing is a critical activity, it has proven more effective to have a separate testing team instead of expecting designers to also test. (Yetton, 1983)

Recommended Core Techniques

- **Time-boxing:** Milestones are used in traditional project management to agree on a deliverable by a given time. In the same manner as milestones, DSDM uses a core technique for scheduling delivery of work known as Time-boxing. Time-

boxing is basically an interval that is no longer than 6 weeks where a set of tasks must be completed. It has been proven multiple times in project management that estimates for short duration efforts are far more accurate than long duration efforts In short, time-boxes consists of multiple tasks and the project is made up of multiple time-boxes. Having this fixed deliverable with the use of time-boxing supports the concept of milestones. However, the time-boxes are subjected to change. While the tasks are defined, the priority of the tasks is allowed to change in an effort to maintain an environment that can respond quickly to business needs. That said, DSDM supports the reduction of functionality in favor of making delivery on time.

A few of the attributes of time-boxing are listed below.

- Time-box length can change.
- Execution of time-boxes in parallel is permitted.
- There can also possibility be nested time-boxes.

Finally, while DSDM places emphasis on schedule over scope we live in an imperfect world that time-boxes can't necessarily fix. There will be some cases when management agrees that the cost of a schedule overrun is acceptable when a "must have" features appears.

- **MoSCoW Rules:** As stated before, since DSDM projects need to be focused on both time and cost, there is a high level of engagement with users during the development process. This means for customer satisfaction it is always necessary to keep an eye on the features that are the most important to the user. There are generally two main reasons a user decides to make changes in a project. The first is that it is possible they learned of new technologies that they can take advantage of and want to use. Second, is that their work environment and business needs have possibly changed. In both cases a swift re-evaluation of priorities is needed. Swift in this case means easy to understand, to the point, and organizes features into different groups. To prioritize the options, DSDM supports the implementation of MoSCoW rules.

Must Have

All features that are listed in this group must be implemented. These are the showstoppers. If they are not completed then the product won't meet requirements.

Should Have

The features that are listed in this group are also important and contribute to the value of the end product. However, they are not mandatory and if time constraints need to be met then they can be omitted.

Could Have

These items generally represent enhancements to the finished product and these features could easily be added later.

Want to Have

This category of features represents the items of least importance. Usually these features serve a limited user group and they are of little value.

- **Prototyping:** The introduction of prototyping in DSDM projects fulfils two DSDM principles. They are the principles of frequent delivery and incremental development. The purpose of prototypes is to validate critical functionality in the early stages of the development process. They also help to get user feedback on early deliverables. The incremental nature of this process permits the early enhancement of a product based on relevant user feedback. DSDM differentiates prototypes as listed below.
    - Business Prototype that allows evaluation of the system.
    - Usability Prototype used to check the user interface.
    - Performance or Capacity Prototype that ensures the solution will deliver the required performance.
    - Capacity or Technique Prototype to assess possible options.

- **Facilitated Workshop:** With many project life cycles workshops are used in order to identify and define development tools that will support user-developer association. However, as projects become larger a few problems occur. Many times large, heterogeneous teams and groups can create either a "Group Think" situation or "Analysis Paralysis." DSDM strives to avoid these issues by limiting the number of participants for a workshop. There are 5 common workshops in the DSDM life cycle and they are listed below.
    - Information System Requirement Definition.
    - Information Benefits for Business.
    - Technical System Operations.
    - Planning of Acceptance Test.
    - Information System Design.

Critical Success Factors

There are several key areas that need to be managed throughout the DSDM life cycle in order to achieve project success. The critical success factors compiled by the DSDM consortium are listed below.

- Acceptance of the DSDM philosophy before beginning work.
- Defined decision authority of users and developers on the team.
- The commitment from senior management to acquire end-user involvement.
- Incremental deliveries.
- Easy access to users by developers.
- Stable team.
- Skilled development team.
- Size of the development team.
- Supportive customer supplier relationship.

- The development technology.

The above list is extensive, however, it is worthwhile to draw attention to the fact that the basis of success in DSDM projects is strong leadership and executive management support. DSDM demands an advanced corporate culture in order to achieve success. That said it is possible that some of the critical success factors listed might not apply to all projects (i.e. supportive commercial relationships may not be possible with open source projects).

Related Methods and Quality

Compatibility Maturity Model and ISO

While there is no guarantee that DSDM can enhance a company's CMM process, the introduction of DSDM would help a company move from CMM level 1 to CMM level 2 by implementing a repeatable project management life cycle. This is also true for processes and methodologies like ISO.

Extreme Programming

This was the first well known methodology to handle Agile software development. As a member of the Agile family of life cycles, it is perfect for integration with DSDM because many attributes of DSDM can improve the robustness of XP.

As mentioned in this chapter, the framework of DSDM is built on project management best practices. While DSDM is not perfect for all projects, its strengths are ease of use and flexibility. With this in mind, a team should still be cautious when using DSDM with XP because even the creators of DSDM warn about its use on all projects.

One of DSDM's weaknesses is, as with other structured approaches, the relatively large barrier to entry. Moving to DSDM is neither cheap nor fast and requires a cultural shift in any organization. This cultural shift is largely because DSDM support scope changes in order to deliver on time and within budget. For many people accepting that fact can be quite difficult.

DSDM is generally considered the most flexible Agile development method because it can be used with any project whereas many Agile methodologies are mere software development life cycles. That said, a few Agile methodologies do share a common base with DSDM. SCRUM for example promotes team empowerment much like DSDM. Additionally,

Crystal embraces a pool of best practices. As DSDM continues to evolve we can expect that revision control of the release will be continued by the DSDM consortium.

DSDM Projects

The primary use of DSDM is software development. However, because of its flexibility, DSDM can be used in many diverse projects. The attributes that allow for this wide range of project management environments are listed below.

- DSDM is a general framework for solving complex problems.
- It breaks down business requirements into small understandable user stories.
- Assesses business needs and prioritizes the requirements according to them.
- All deliveries, acceptance, and testing is completed in a small time frame.
- Delivery is completed with a team that includes the users.
- It drives projects toward completion within time and cost constraints.

Advantages of DSDM

There are number of advantages of DSDM and they are listed below.

- Development results are immediately visible.
- Greater chance of user acceptance because of their involvement.
- Basic functionality is delivered quickly thereby creating project efficiencies.
- Streamlined effective communication.
- Decision to use DSDM is made at the beginning of the project.
- Users and the development team are empowered to make decisions.
- Greater success rate due to continuous feedback of users.
- Focused on business requirements.
- Effective collaboration is encouraged between all parties.

- Every deliverable is built on a strong foundation and incrementally released.
- Constant communication ensures that everyone understands what is going on.
- Project controls ensure there is no compromise on quality.

Disadvantages of DSDM

Along with advantages of DSDM there are also disadvantages that are listed below.

- Cost of licensing.
- Relatively high barrier to entry.
- Organizational culture shift.
- DSDM is not good at marketing.

Chapter 10

Agile: Lean Development

Lean Development focuses on the production of the maximum amount of value with the minimum amount of work (Vajda, 2010). It leverages the best in delivery teams and optimizes their efforts for success. Delivery teams all across the world have benefitted from the tried and true principles of this methodology.

What is Lean Development?

Lean Development is an amalgam of lean manufacturing and Information Technology principles. This concept, as deployed in software development, makes up the core of the Lean Development life cycle methodology. It is akin to manufacturing and production and derives its principles from these disciplines.

The fundamental set of activities that are involved in the lean process include solicitation of requirements, documenting specifications, design of the architecture, planning the software build, design, testing of the product, debug, deployment, and maintenance (Poppendieck & Poppendieck, 2003).

History of Lean Development

The Lean Development life cycle first came to the forefront during 2003 in the book titled “Lean Software Development: An Agile Toolkit”, written by Mary Poppendieck and Tom Poppendieck. This book laid out the foundation of lean along with their modifications. It also listed 22 tools along with their comparison with their Agile counterparts (Poppendieck & Poppendieck, 2003). In addition to their book, the authors went the extra mile to explain and explore the concepts of Lean Development through conferences and other avenues.

It has been said that the basics of Lean Development evolved from a Japanese auto manufacturing company named Toyota that had adapted to Agile principles (Monden, 1998). Toyota developed a comprehensive system in the 1970s (Peeters, 2013). In the 1980s Toyota was the pioneer in implementing the principles of Lean Development (Vajda, 2010). Toyota claims to have decreased their production times by as much as 30 to 50% by implementing this methodology (Morgan).

Previously named the Toyota Production System (TPS) the method later became known as Lean Development and the company progressed swiftly by implementing the key principles of this life cycle model. Toyota ultimately reached the pinnacle of their capacity with an emphasis on producing several models of automobiles from varied platforms. This enabled them to fully use their human potential, minimize waste and get the most efficient output. A quality engineer by the name of John Krafcik was the first person to use the term Lean Development for the procedures that they employed at Toyota Motors (Csaba, 2013). Today, every automobile manufacturing company in the USA uses Lean Development to get this production edge. However, Toyota is not the only organization that found its way to success using the Lean Development methodology. Other infamous names on the list of successes include Wal-Mart and Dell (Nallasenapathi, 2006). Furthermore, this is a life cycle that is finding its way into many industries that strive for the absolute best results.

Taiichi Ohno, who laid the foundation for Just-In-Time Manufacturing (JIT), emphasized that cost reductions should be the basic premise of all operations. This turned out to be the first principle of Lean Development because by reducing waste costs were reduced.

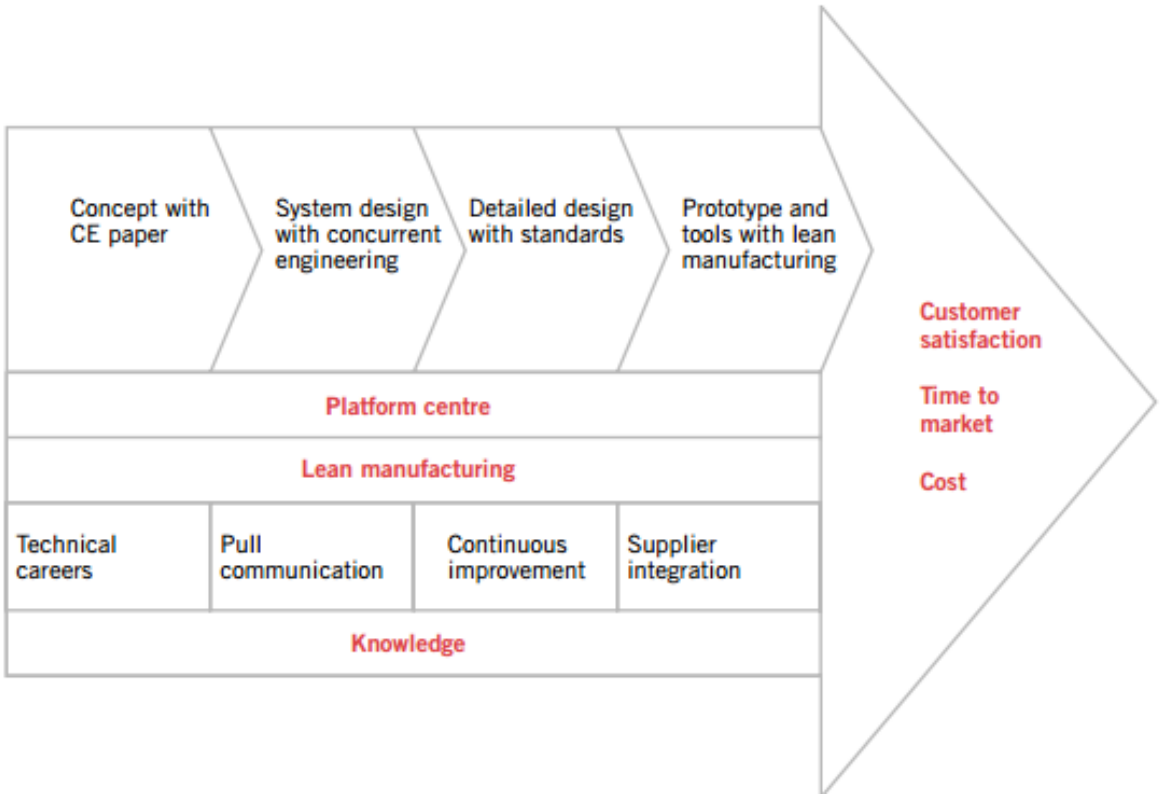


Figure 10.1 Toyota's Implementation of Lean Development

Lean Development Principles

There is a set of seven fundamental principles that make up the core of this life cycle. Implementing these Lean Development principles leads to what many would consider a "good enough" solution (80% good) that satisfies the customer requirements. Lean Development does not aim for perfection (100%) because it seeks to deliver the most efficient and effective results utilizing all resources in the best possible manner. These basic principles are discussed below.

Principle 1: Eliminate Waste

According to Lean Development principles everything should exist for the benefit of the end customer. If the end user does not make use of any attribute of the product it is "waste." This philosophy places emphasis on getting rid of every aspect of waste in the final product, whether that waste is in code development, project management, or even the effort and requirements for individual team members.

The most important thing for the delivery team is to spot what may be characterized as waste. A method called "value stream mapping" is used to determine whether a particular aspect of a project is waste or not. The simple truth with this life cycle is that success of a project depends only on whether the end customer is satisfied or not because in Lean Development the highest priority is placed on customer satisfaction (Nallasenapathi, 2006). All team members need to put themselves in the end users' shoes in considering whether or not a particular aspect of a project is adding value for the end user or not. Anything that a customer would not like to pay for is not of value (Vajda, 2010).

Any redundant activity that can easily be skipped without causing negative impact on product quality is skipped because the redundant activity is not benefitting the end user. Bureaucratic activities are also considered waste. Since an efficient process is what is desired, bureaucracy has no place on the project. In any Agile project, more than bare necessities of documentation and the time spent waiting for information from others in the communication process is also considered a waste (Vajda, 2010). In a manufacturing environment like Toyota wastes can be in the form of defects, over processing, over production, waiting, motion, inventory, transport, cost of poor quality, and other more (Morgan). Toyota is a Japanese company and in Japan waste is termed as "Muda." This term has been used many

times in literature to describe the concept of waste as it is referred to within this life cycle methodology.

Additionally, Lean Development principles state that any development procedure that was left incomplete and did not add any significant value to the final product is the delivery team's fault and is not useful for the consumer. Thus it is waste as well. Anything that is not in accordance with a user's needs may also be eliminated. Additionally, substandard quality should be eliminated. For example, superfluous comments written within the code are also a form of waste for the consumer if they do not add to their learning experience.

As with many Agile methods, red tape is the enemy. Any slowdowns created by red tape on the part of the development team are not of concern to users. This drives the elimination of any overhead from the developers. Of course, in order to get rid of waste, it must first be identified so it is important to monitor who and what is contributing to waste. Getting rid of the waste in any project should be executed systematically until all team members are sure that every instance of waste has been eradicated.

In order to implement Lean Development effectively, the delivery team must vigilantly promote this mode of thinking where they constantly ask themselves at every step the rationale behind their actions. The elimination of waste is not a one-time effort. It is instead a continual activity and a practiced routine that becomes second nature to all members of the project team.

Principle 2: Build Quality In

The customer experience of the end result should be exactly what they expected or even better. Additionally, the customer's confidence in the ability of the solution to deliver as expected reliably will increase if they have a complete understanding of the final result. To facilitate the customer's confidence, Lean Development emphasizes face to face communication between the end user and the project team. This transfer of information (TOI) is not a onetime process but instead should be repeated as often as feasible to make sure that both parties are on the same page. The Poppendiecks refer to this as "respecting people" so that they have a sense of ownership and feel a tie to the success of the project.

Lean Development principles encourage the delivery team to look at the whole picture and work on everything. This is contradictory to a staged or phased approach where everything proceeds linearly and a solution is completed in steps. The preference of direct

communication over written specifications tends to yield a more clear and precise understanding of the requirements. The code developed in Lean Development is kept clean and free from repetitions. Automated tests and other detailed testing procedures should be carried out before delivering the end product, or any incremental release, to make sure that there are no defects. Functional testing ensures that the intended purpose is fulfilled and acceptance testing guarantees customers are satisfied with the product delivered (Vajda, 2010). Testing procedures are also aimed at decreasing any instances of waste.

Principle 3: Create Knowledge

Developing a software product is not a static process and it can only be executed effectively if the team embraces a continuous learning process in order to enhance their skills and improve the final product. In order to make any product a success it must be aligned with user requirements and every effort should be made to make sure that those user requirements are completely understood. Tactics such as reviewing screenshots may also help the project team communicate their understanding of requirements and make adjustments as necessary.

Essentially, Lean Development emphasizes that customer satisfaction can only be achieved with a process that includes continuous training for the project team so that the final product delivered is of the best quality. In handling product challenges, if something does not work after several attempts then it is practice to move on and waste no more time on the effort. Instead, the delivery team should opt for alternative ideas. As such, the principal is that the more the team members receive training, the better they are prepared to manage risk (Jailia, Mrs.Sujata, Jailia, & Agarwal, July, 2011). Training, however, is not limited to the project team but also includes the customer whose knowledge must be increased at every level of the effort. Increased knowledge paves the way for better risk management. This knowledge also helps drive identification of critical requirements as well as unnecessary ones.

In carrying forward the effort of understanding requirements, the delivery team should strive to uncover and fix all code defects as soon as possible. Furthermore, if code is compiled without performing testing and debug, many defects will be missed and affect the quality of the final product. Once the progressive elaboration of requirements definition has been completed and all requirements are clear to every team member, a transfer of information has taken place (Vajda, 2010). To support this end, it is important to increase understanding and learning through effective communication in every meeting that identifies

all project limitations to make sure that everyone understands the inherent constraints. With better knowledge of project constraints, come better solutions. As such, another effective communication tool for product delivery is the use of demos. Demos can be used in meetings to give users an idea of the look and feel of the final product.

As part of their Lean Development principles, Toyota Motors also spends considerable time on learning before working so that the knowledge can benefit them later in the project. In fact, Toyota boasts of their commitment to learning. They conduct concurrent and post-mortem lessons learned events to pursue continuous improvement. Knowledge acquisition is further facilitated by reducing the development time through incremental releases consistent with Agile. In this way, customer feedback can be incorporated in each version of the product.

When the end user and delivery team sit together in meetings to discuss each incremental release of the product, new possibilities emerge and learning increases on both sides. Involving customer communication at every step of the project is good leverage in this regard. Real time feedback always delivers the best results (Vajda, 2010).

Principle 4: Defer Commitment

As with other Agile life cycles, Lean Development expects changes. Important decisions made early in the project are likely to change in later stages of the project. With Agile, the best choice for software development is to delay critical decisions until the later stages of the project when the team is more aware of all the impacts of potential changes and the direction development is taking. That said, postponing irreversible decisions is a good idea as long as the delivery team is sure that they are headed in the right direction.

Deferring decisions, of course, simply means that they will remain open until a later time. At that later time, the decision makers will have more information regarding their options and will be in a better position to choose a path. However, there are potential downsides to deferring decisions. It is quite possible that this tactic can go very wrong if decisions are not made at the right time. Examples of changes that can be accommodated with late decisions include market changes, technological advancements, and requirement amendments.

Still, Lean Development principles are based on the premise that decisions that are delayed until the later stages of the project will likely be better decisions. All too frequently,

decisions made earlier have to be changed to address unforeseen issues. Subsequently, the less often critical decisions need to change, the lower the cost of the end product.

The Poppendiecks have termed the latest time that decisions can be deferred to as "the last responsible moment." Essentially, if the various phases of delivery are not tightly coupled, it should be easier to carry out execution in any order. As knowledge increases with every incremental release of the product, according to Lean Development principle number three, the developers are in a better position to make informed decisions based on progressive elaboration during the course of the project.

History has shown that delaying these decisions also allows the customer to keep their requirements in sight as well as the options that are available to them. Required delays in decisions are especially beneficial for the technological decisions. Right from the beginning of the project proper planning should be completed to allow for the flexibility in making such decisions.

Principle 5: Deliver Fast

Technology continues to evolve at a rapid pace and newly released products become obsolete quickly. Thus it is of utmost importance to the project team to avoid any unnecessary delays in the delivery process. In short, the time to market for the product must be reduced to bare minimum time. This does not imply delivering an early product release that is not in accordance with the user's needs, but instead stresses delivery of a quality product without delay. Essentially, the focus is on efficient execution the first time, but leaving margin for changes.

The faster a bug-free version of the product can be delivered, the more rapidly feedback will be received allowing the project team to incorporate necessary changes. Also, feedback may highlight features that are not required by the end user and thus may be eliminated in the future to reduce waste from maintaining undesired functionality. In order to deliver follow up increments quickly, the same team that produced the previous version of the product is required to update it. If outside resources are employed to fix problems, they must take the time to first familiarize themselves with the project before they can provide assistance.

Delivering rapidly is not in contradiction with Lean Development principles of delaying decisions. In fact, when the two principles are combined, work is delivered as fast as

possible. Decisions are delayed as long as they can be and the project reaps the full benefits of the Lean Development life cycle. Getting the right people to work with the right mindset will help achieve these dual objectives.

If changes are to be integrated into the design, they should be completed without making changes in functionality and vice versa. This strategy not only saves time but also reduces the chances of error and waste. Time is money, and this is very true in the world of project management. As with other Agile life cycles, shorter incremental releases enhance communication with the end customer thereby improving project performance.

Another effective practice for efficient project delivery is to allocate more than one team to a specific project so that they can learn from each other. Multiple teams can provide multiple solutions to a problem and if any one solution is not appropriate for the issue at hand, it can be disregarded. A meeting with the teams can be held to easily choose one solution. History has shown that delivery of incremental releases also tends to divide the risk and make it more manageable. In determining the frequency of incremental releases, a cost analysis may be of benefit (Vajda, 2010). Release cycles generally depend on project size and the delivery team's preference but basically anything between two weeks and four weeks is optimal.

Principle 6: Respect People

The Lean Development life cycle principles state that the project team should be empowered to perform their tasks with a sense of authority and responsibility. This approach tends to enhance creativity and delivers superior results. Two-way feedback between the team members and project manager is essential to ensure empowerment of the project team (Vajda, 2010). All members of the project team should add value and, as such, empowerment often motivates them to continue providing good work.

Historical views of project management dictated that the project manager should oversee all work of the team and "tell them what to do." In opposition to this thinking, Lean Development takes a path that stresses empowering the team so that they can complete their tasks without constraints. The project manager should establish a clear direction and align all project team members. Once this direction has been communicated, everyone is encouraged and empowered to pursue their own methods for completion of their tasks.

Respect for other people also supports the core concept that everyone should be given the liberty to work on their part of the project and make effective decisions within the limits of their authority. Of course, the project manager has to still review all major decisions, but if decision making is decentralized to a certain extent, it will empower every member of the project team.

Theodore Roosevelt is worth mentioning in this regard. He was once quoted as saying "The best executive is one who has sense enough to pick good people to do what he wants done, and enough self-restraint to keep from meddling with them while they do it." The same is true for a project manager.

For example, if developers are only seen as resources who are supposed to write code, as and when required, without being given room for creativity, then they will naturally feel less motivated. Lean Development embraces the concept that in order to ensure success it is imperative to hire the best people and let them do their magic. The project manager's prime responsibility is to encourage the team members and support them in completing their tasks. Of course, he also needs to make sure that everything goes as planned and any unforeseen circumstances are dealt with promptly. A sense of ownership is always an element of high performing teams.

Principle 7: Optimize the Whole

Successful project management using this life cycle calls for the implementation of all Lean Development principles. The delivery team needs to have an overview of the whole solution but still divide it into smaller manageable parts to tackle each element separately. With each customer interaction, more and more discrepancies surface, thus prompting the learning process and improving the overall results. At every stage of the project every stakeholder should be involved to ensure complete participation and utmost satisfaction.

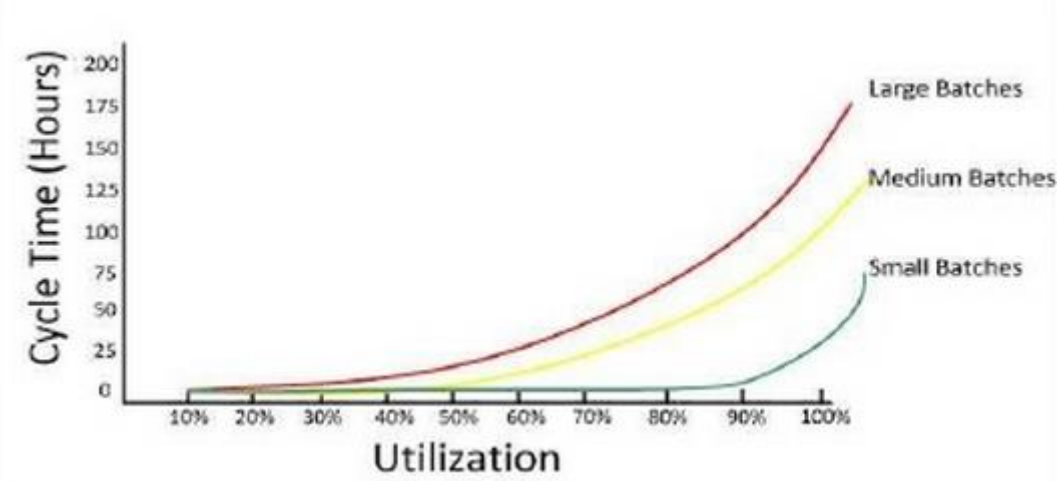


Figure 10.2 Balance Between Releases and Utilization

An organization implementing Lean Development should strive to optimize the whole project and integrate every aspect of the life cycle into operations. This brings uniformity to the end result and the project will be cohesive. Additionally, if everyone strives to manage to the bigger picture it will result in better coordination and communication between the different teams working on the project.

Tools of Lean Development

The principles of Lean Development as listed above were implemented by the Poppendiecks (Poppendieck & Poppendieck, 2003). Before executing any project using the Lean Development life cycle, all team members should be clear about each tool or principle involved with the Agile methodology. These seven principles are the best practices and they are executed by using the twenty-two tools of Lean Development (Vajda, 2010). Other Agile methodologies have similar tools that are used in conjunction with their procedures.

Lean Software Development

The Lean Development life cycle has its roots firmly planted in the automobile industry, but today it has found widespread use in software development as well. The software industry has benefitted from this life cycle in the form of improved processes, resolution of inefficiencies, and superior results among other associated areas.

Waste in software development processes can be in the form of redundant features, extra steps in code, repetition, time wasted in finding information, waiting for feedback on decisions, handoffs and more.

Advantages of Lean Development

There are several advantages of Lean Development for the project team and other stakeholders. Some of them are listed below.

- Small batches in Lean Development result in an efficient system (Vajda, 2010).
- Options and open communication lead to favorable results.
- If the project's priorities change over time, Lean Development can easily accommodate them with incremental product releases.
- Lean and Agile are blended well with this life cycle. Thus the benefits of Agile can be reaped while using Lean Development (Woods, 2010).
- Like other Agile methodologies, Lean Development supports communication over documentation with the end result being clearly understood requirements.
- Shorter development cycles increase commitment within the project team. Each incremental product release boosts their motivation and grows their drive to complete the project even faster.
- Small incremental releases make it easier to identify and eliminate bugs.
- Since emphasis is placed on development of one product feature at a time Lean Development ensures that dedicated efforts are devoted to each and every part of the project (Jailia, Mrs.Sujata, Jailia, & Agarwal, July, 2011).
- Increased feedback in the Lean Development life cycle ensures that the customer has a say in every aspect of the final product and that it is in complete accordance with the requirements.
- Rapid delivery of work enhances the end customer's confidence in the project and the team's abilities. If the solution is delivered early, there is little chance that the consumer will have an occasion to change their mind (Vajda, 2010).
- The features that contribute the greatest amount of value can be developed early in the project.

- Application of Lean Development principles during delivery and manufacturing stages provides maximum effectiveness (Morgan).
- Delaying decisions often increases manageability of change. When important decisions are made late in the project, when knowledge levels have increased, changes can usually be easily incorporated (Vajda, 2010).
- Lean Development practices tend to avoid critical mistakes where decisions are based on assumptions. Typically, decisions within the Lean Development process are made with solid facts in hand. Important decisions are made only when requirements are completely clear to the project team.
- Lean Development transforms an organization into an event-based entity where something is only manufactured or produced when an order is received from a customer.
- The Lean Development methodology can reduce inventory costs and redundant products that end up being left unused (Woods, 2010). Producing only when a customer requires has been termed as a Pull System.
- A mindset of continuous improvement allows a project and an organization to keep up with the pace of evolving technologies and emerging trends.
- In times of recession, products manufactured using Lean Development will have a big edge as they will be delivered quickly, managed easily, resistant to risks, and financially sound (Jailia, Mrs.Sujata, Jailia, & Agarwal, July, 2011).

Disadvantages of Lean Development

There are equally several disadvantages of Lean Development for the project team and other stakeholders. Some of them are listed below.

- A high level of communication is required for successful execution of Lean Development principles. Any breakdown in communication among stakeholders can create project failure and nullify the principles being implemented and the full project benefits being sought.

- Delaying decisions for as long as possible can develop a mindset where commitment is avoided.
- Uncertainty prevails because any of the available alternative solutions can be opted for at any time.
- If the delivery team or the project manager fails to recognize the point of no return for a pending decision, the project can become a sheer disaster (Vajda, 2010).
- If incremental deliveries are not consistent the end user can become easily distressed. So, it is important to set realistic deadlines for each delivery and meet them.
- Many people have a problem transitioning from other methodologies to Lean Development because of the many principles and tools that Lean Development utilizes.
- It is important to realize that implementing Lean Development may not be the feasible choice in every case. Thus, if pushed beyond its limits as a life cycle, resources could be wasted (Nallasenapathi, 2006).

Chapter 11

Agile: Feature Driven Development (FDD)

The world of software development life cycles seems to continue to evolve rapidly. In an effort of continuous process improvement newer and better methods are emerging that bring great benefits for developers and end users alike. Feature Driven Development is an Agile life cycle that is both iterative as well as incremental. This ultimately means it combines the benefits of both. The best practices of Agile are once again merged into this methodology. Hence, this is a life cycle that is preferred by a great majority of development professionals worldwide. This model is generally known by the acronym FDD.

History of Feature Driven Development

Jeff De Luca was a professional working in a Singapore bank. He was assigned to a software development project that spanned over a period of 15 months and encompassed the skills of some 50 people. Working on this project, he eventually developed his own methodology that later evolved into the Feature Driven Development (FDD) life cycle. Thus FDD came to existence in 1997. Peter Coad lent his approach on object modeling that later became the foundation of FDD. Since the Singapore bank project became a widely known success, many other projects opted for using the life cycle. When the first project was successfully completed using the principles of FDD, another project was initiated that lasted 18 months and involved 250 people.

After the successes in 1997, FDD principles were documented in a book. In 1999 three authors, Eric Lefebvre, Jeff De Luca and Peter Coad, combined their expertise and experience on the effort and started writing “Java Modeling in Color with UML.” Feature Driven Development was detailed in Chapter 6 of this book. It was published in 2002 and provided a general description of FDD. Peter Coad showed that a combination of modeling in color and Feature Driven Development can work extremely well for large projects.

Even today, the founder of FDD, Jeff De Luca, has a company by the name of Nebulon dedicated to building a community for learning, growing, and sharing experiences regarding Feature Driven Development (Palmer & Felsing, 2002). They also offer training courses and workshops to expand the learning and implementation of the methodology. Additionally, certification is also offered to recognize true professionals of the life cycle

model. If at any time someone has any questions, they are always able to go back to the community website to find the answers they need.

What is Feature Driven Development?

Feature Driven Development is an amalgam of model driven approaches and Agile techniques. Thus it combines the best aspects of both. It has been classified as a light weight life cycle for software development that executes by way of incremental releases. Today, this software development methodology is being implemented in all business domains. It is especially effective for working on large projects or managing work among large teams and, once again with Agile, the blend of iterative and incremental approaches makes working on large projects easier. FDD is centered on features that can be defined as manageable, client-valued functions. Feature Driven Development works on the principles of domain object modeling and UML in color.

FDD leverages a set of five processes that manage the overall software development project. While the first process was based on Peter Coad's approach to object modeling (Develop an Overall Model), the second one makes use of a features list. The last three activities of FDD starting with Plan by Feature are repeated over and over again to develop each feature of the software product. Design by Feature and Build by Feature are performed incrementally until all the processes have been executed. Almost three fourths of the work completed on the project is performed during these two steps. This life cycle development methodology makes use of short cycles of incremental releases covering all these five processes. Furthermore, in line with project management best practices, milestones are created to manage project progress.

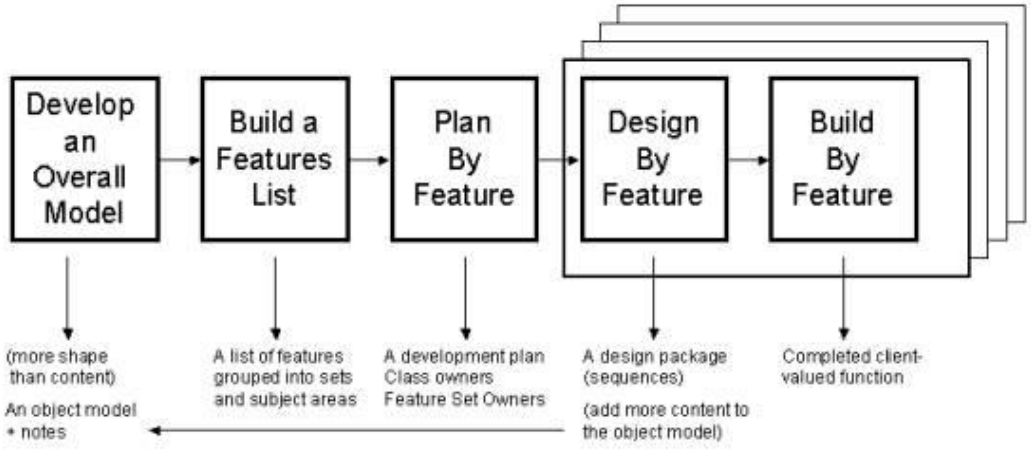


Figure 11.1 FDD Processes

Structure of a Feature Driven Development Project Team

Working on a Feature Driven Development project requires a team that truly understands the activities involved. Without complete understanding of team responsibilities the project cannot be a success. The positions that perform major tasks on a FDD project include a Domain Expert, a Class Owner, a Chief Programmer, a Development Manager, a Chief Architect and the Project Manager. As with other roles in life cycle processes, several of these roles can be performed by multiple people at a time so that developing different features of the software is more efficient. On the other hand, a single person can assume more than one responsibility on the development team as well.

In addition to the key roles there are supporting roles on the FDD team. These include Technical Writer, Deployer, Tester, System Administrator, Toolsmith, Build Engineer, Language Guru, Release Manager, Domain Manager and possibly others. Since Feature Driven Development can be used for large projects with large teams, there could be other roles defined according to the needs and scope of the project.

Further detail on the defined roles for a delivery team using Feature Driven Development is below.

Project Manager

The Project Manager for an FDD project is responsible for overall management of the project including major responsibilities like keeping an eye on the progress of the project, establishing budget allocations, managing headcount, monitoring equipment usage, maximizing use of work space, and controlling all the other available resources for the project.

Chief Architect

The Chief Architect of any FDD project works on all the design aspects of the project. As the development team works on creating the design for the project, the Chief Architect oversees their work. Most of the technical details regarding the design and its implementation are handled by this key role.

Development Manager

Development is the core activity of any Feature Driven Development project. With that in mind, the Development Manager is the person in charge of all the development work initiated on the project. He is also responsible for maintaining unity and cohesion between the working team and resolves any disagreements that arise during the development process.

Chief Programmer

The Chief Programmer has a bit of an edge over all the other programmers in that he is well experienced with the processes involved with Feature Driven Development projects and their intricacies. They have a complete skillset with regard to analysis, design and delivery of software projects. A Chief Programmer coordinates the work of a number of other programmers or developers. This number may vary between three and six.

Class Owners

Class Owners work under the guidance of the Chief Programmer on various activities involved in a FDD project. Class Owners have complete accountability for the part of the job (project) that they work on. This is to say that they "own" the class that they work on. From designing to coding and from testing to documentation, all aspects of the class are handled by the Class Owners.

Domain Expert

A Domain Expert can be anyone who has thorough knowledge of the system being built and someone who is in a position to help the developers understand the intricacies of the project. It is important to have effective communication among the Domain Experts and the delivery team so that everyone is headed in the right direction.

Domain Manager

A Domain Manager is someone who coordinates the work efforts of Domain Experts. Every Domain Expert gives the input of their expertise based on the situation at hand. The essential responsibility assumed by the Domain Manager is to coordinate communication of all input received from Domain Experts to the delivery team to make sure that the input is incorporated into the delivery of the product.

Release Manager

Regular meetings are established between the Chief Programmer and the Release Manager. In this situation, the Release Manager gets input on work progress from the Chief Programmer about how far the project has progressed and then communicates the progress to the Project Manager. Thus, he or she works on controlling the project progress and compiling the information regarding it.

Language Guru

The Language Guru is someone who ensures that all the details of the programming language are well understood and duly implemented with the FDD project. If the development team is adept with working with the programming language, it will not be a big deal. However, if the development team is still learning the complexities of a programming language, the whole development team is relying on the expertise of the Language Guru. That said, it is not mandatory to have a separate person as the Language Guru, because any Domain Expert, Class Owner, Chief Programmer, or any other qualified person can fulfill the need.

Build Engineer

The build process is yet another core activity of any Feature Driven Development software project. The Build Engineer handles the maintenance and implementation of the build process and coordinates with other stakeholders on the delivery team.

Toolsmith

The Toolsmith is a versatile team member who develops a number of tools that the engineers use in the delivery of their work. These tools are generally small and can be used for development or testing purposes.

System Administrator

This is the person who manages the computer network the members of the delivery team use to communicate, coordinate, and share their work with each other. The System Administrator handles everything from setting up the network to managing it. If any problems occur with the network this role is responsible for troubleshooting the issue, resolving the problem, and ensuring the FDD project continues to run smoothly.

Testers

Testers may be specialized independent verifiers of the code or some of the developers may double as Testers based on the size of the project. They bear the responsibility to make sure that the code created performs the intended functions and there are no errors. All the system functionality is matched against the user requirements by the Testers in a Feature Driven Development project.

Deployers

If data conversion is required from one format to another, this task is performed by the Deployers for a Feature Driven Development project. Whenever any version is released for the software effort, Deployers implement and physically deploy each version and make sure that everything goes smoothly.

Technical Writers

Feature Driven Development does not require extensive documentation for everything on the project but it still supports the need for appropriate user documentation to be available as and when required. If no one else on the delivery team is already writing documentation, a special technical writer may be hired to perform the function or any one of the other team members can be assigned the task as needed.

The Five Basic Activities

As mentioned earlier, Feature Driven Development is based on a set of five basic activities that govern the overall delivery of software products and project management. XP works on the concept that before actually executing the project plan, an iteration zero is conducted and the delivery team gathers everything they require to start working. FDD, on the other hand, does not believe in any iteration zero and starts working on the five processes immediately. Each FDD process can be summed up on two pages (Palmer S. , 2009). When each activity and process is well defined, it is easier to engage new team members because the ground work has been completed and allows for learning time to be reduced. The activities (or processes) performed during FDD projects are then broken down further into sub-activities to make the implementation more manageable.

Develop Overall Model

When the actual work starts on a software development project, models of various domains are developed and later combined to develop a complete project model. However, prior to this effort, it is important to clearly understand and define the scope and context of the project so that all aspects are clear to the team. Small groups build models for each domain. The proposed models are then compared and the best one is selected. Sometimes models are combined to achieve the optimal design.

After completion of the outline, the delivery team starts implementing the general model instead of focusing their attention on formal documentation (Palmer S. , 2009). Design and planning documentation is very limited in Feature Driven Development (FDD). This is because developers normally prefer small summaries over lengthy documents. As stated above, once a consensus is reached on the domain models, all the models are rolled up to complete the full model for the project. If any adjustments are needed because of combining models, these modifications are made as the project progresses. This effort of developing the plan is incremental and performed with the collaboration of the whole team.

Some team members who want to bypass this first step might argue that it is useless but as history has shown, developing an outline for a project makes sure that everyone on the delivery team is on the same page. The first tasks ensure that all team members completely understand the project scope and its constraints as well as the resources and their availability. This includes the entities involved and the nature of their relationships. The broad understanding of the project makes it is easier for the developers to move forward and work together efficiently.

Build a Features List

The list of features to be delivered by the FDD team is developed using information gained during the construction of the overall model for the project. Once the small groups have made their domain models, they can use the same knowledge to create a list of features. All that the sub-teams have to do is to divide the domain areas into even smaller subject areas so that manageable lists can be produced. In short, the business activities contained within each subject area make up the content of the feature list. Features are manageable work packages that take up to a maximum of two weeks for completion. If for any reason they span beyond this timeframe, they are decomposed even further to drive successful completion.

Having new tasks to work on every two weeks also helps keep people motivated and energized. While this two week timeframe is a maximum, usually the standard time frame for a feature's completion is 1 to 5 days (Palmer S. , 2009). The feature list is in fact managed using a three tier hierarchy. Each functional area is decomposed into activities and features. Additionally, the sequence of the feature list can be changed as and when required. As the name of this methodology implies, everything is going to be driven based on these features. The feature list is usually expressed with a structure that is <action>, <result>, <object> to build functions that deliver value to clients.

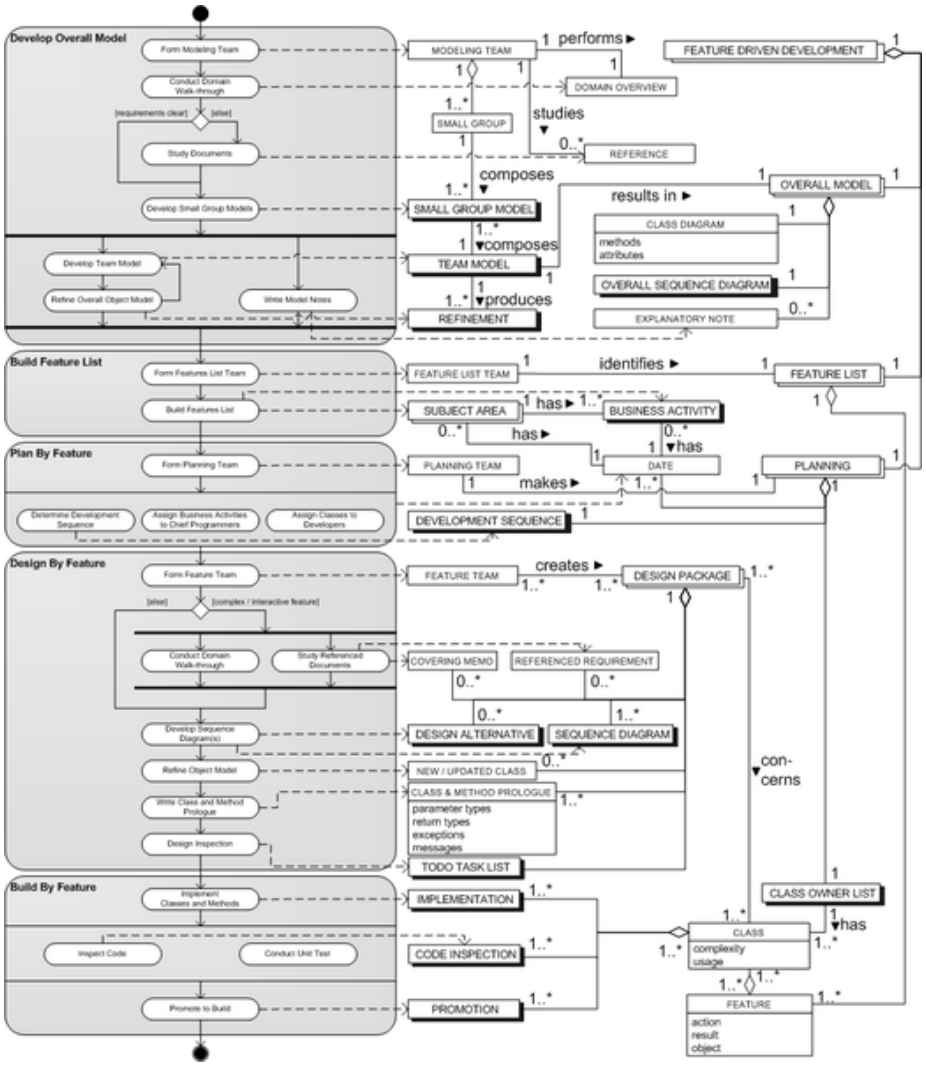


Figure 11.2 Figure Shows Process Data Diagrams in Feature Driven Development

Plan by Feature

Once the feature list is ready and available, a development plan is produced based on the list. Different features, or sets composed of various features, are ordered as classes. Each

of these classes is then assigned to senior programmers so that everyone knows what is to be accomplished. These assignments ultimately define the class ownership for every feature class. In other Agile software development methodologies, the code belongs to everyone and everyone is equally responsible. However, in Feature Driven Development, class ownership assigns sections of code to specific programmers. All of this planning and assigning of classes is completed based on the outline that was constructed in earlier project activities.

Design by Feature

The same feature list is used to agree on a design package. Every senior programmer takes responsibility for features from their assigned list that can be managed within the two week timeframe. Afterwards sequence diagrams are made for every feature. These sequence diagrams are then merged to develop a complete model for the full project. When development is complete, tests are executed to make sure that features perform as designed.

A design package for any Feature Driven Development project will typically include a document that explains the whole design package and any other supporting documentation. Sequence diagrams and design alternatives are also included in the package. There may also be simple and general “to-do” tasks for every team member working on classes.

Build by Feature

Every feature is to be built with the guideline of client value in mind. This is achieved in building block fashion by using previously tested and passed code as the basis for continued development. Since class ownership is clear from previous activities, every class owner is responsible for successful delivery of the features assigned to him or her. Once coding is complete, unit testing is done to check each section of code. Also a complete code inspection is conducted to make sure that the pieces all perform well together. The final code build is then constructed using these completed and tested features.

Milestones in Feature Driven Development

Feature Driven Development manages workflow on projects using six milestones that are completed in succession. As mentioned previously, features are kept smaller so that they can be managed and completed within the time span of two weeks. The first three milestones from the set of six are completed while the delivery team is working on the overall design from the feature list. The remaining three milestones are completed while the final build

activities are underway. All progress on the project work is carried out on the basis of these milestones. Percentages are assigned to these milestones and used to keep track of project completion. When each milestone is successfully completed, the percentage-complete for the project is increased accordingly. This makes it much easier to report project progress for the delivery team.

Percentage Complete	Milestone
1%	Domain Walkthrough
40%	Design
3%	Design Inspection
45%	Code
10%	Code Inspection
1%	Promote to Build

Table 11.1 Table Shows Allocations of Percentages for each Milestone

Advantages of Feature Driven Development (FDD)

There are a number of advantages of FDD and they are listed below.

- Rapid delivery of the Agile methods and flexibility to adapt with changing requirements.
- Provides a distinct advantage in the form of scalability to an organizational level. (Palmer & Felsing, 2002).
- If outside support is used for code inspection, the fresh perspective can increase the quality of the end product.
- Feature Driven Development (FDD) supports real time measurement of work progress.
- Progress reported by percent complete allows the project manager to more quickly assess project progress. Additionally, these stats increase the ease of stakeholder reporting.
- FDD supports customization and modification so that it can be adjusted according to user requirements (Palmer & Felsing, 2002). This allows any organization to

adapt it to varied project types and still easily implement the process and reap huge benefits.

- Like other Agile methods, there are no rigid deadlines that tend to push developers to sacrifice on quality and simply comply to dates. Tasks and activities are kept streamlined to make sure that there are no unnecessary delays.

- Agile and traditional methodologies both have their own limitations, but Feature Driven Development was designed to overcome most of their shortcomings (Palmer & Felsing, 2002).

- While some project life cycles are designed for large long term projects and others are designed to be flexible to changes, FDD is designed to easily accommodate both.

- Quality is maintained throughout the software development process because Feature Driven Development (FDD) emphasizes code testing.

- UML standards are used for audits and other code elements such as consistency are also reviewed to sustain the absolute best quality standards (Goyal, WS 2007/08).

- The learning curve supported by FDD allows a delivery team to manage their work on the first few incremental releases until they build momentum. Afterwards, the rest of the projects releases are delivered in a consistent fashion. In this way, it becomes increasingly easier during the course of the project to develop features in incremental releases.

- Class diagrams created in Feature Driven Development are more like entity relationship diagrams (ER). This means if a developer understands the ER diagrams, it will be easier for them to adapt to the concepts of FDD.

- Always striving for balance, Feature Driven Development strives to find middle ground between burdensome and lightweight methodologies. In this way, FDD is not too rigid in ceremonial procedures nor too focused on programming and the delivery team can combine the best working practices of FDD.

- Feature Driven Development can be used for complex projects.

- FDD delivers feature by feature in incremental releases to complete the whole project. Additionally, each feature is delivered in two weeks or less.
- The incremental approach of FDD ensures that there is always something that can be delivered to the customers to satisfy them.
- Class ownership lets everyone know exactly what they are supposed to do. If there are any problems or bugs, the team leader knows who is responsible to fix them.
- Everyone bears the responsibility for the feature they developed and making sure that it blends in without compromising consistency of the overall solution.
- Ownership creates a sense of importance so that every programmer can take pride in their work.
- Dividing up the code ownership also provides an added advantage in that incorporating changes becomes easier since no one person needs to wait for somebody else to make changes.
- Like all Agile project management life cycles, FDD prefers the ability to change versus sticking to a plan.
- Testing is executed to make sure that every feature, every class, and every milestone is delivered as expected. Most importantly these tests are conducted both as unit tests, system test, and other inspections as necessary to verify every step of delivery.
- Feature Driven Development is one of the few project management life cycles that is preferred by users, developers, and team leaders alike (Goyal, WS 2007/08).
- Meta modeling within Feature Driven Development adheres to UML standards and compliance with these standards makes any product developed using FDD compact and reliable.

Disadvantages of Feature Driven Development (FDD)

Along with advantages of FDD there are also disadvantages that are listed below.

- Developers that are familiar with methodologies like Scrum and Extreme programming (XP) typically do not like analysis work and can move right through the first FDD processes without giving them proper attention.
- FDD can suffer from many hurdles managing the quality of the work produced, complexity issues, and communication problems.
- It is critically import to hire delivery team members that are good at their work because inefficient workers will make the incremental releases problematic.
- The momentum of an inefficient team will be reduced greatly trying to correct errors repeatedly.
- Since separate FDD teams do not directly interact with each other it can create problems at later stages of the project. Basically, when some teams do not know what other teams are working on or their thought process, it can create consistency issues.
- Normally, the chief programmers on a development team of any Feature Driven Development project do not coordinate with each other regarding the beginning or end of their incremental releases and this lack of communication can develop into problems later in the project.

Chapter 12

Rapid Application Development (RAD)

RAD stands for Rapid Application Development. It is yet another life cycle that is widely used for software development. It uses the least amount of planning methods for the completion of rapid prototyping. The techniques and tools used in this life cycle methodology are basically predefined (Janssen, Rapid Application Development (RAD)). The system revolves around a Graphical User Interface (GUI) that allows users to simply drag and drop their required components on to a page and add functionality. Additionally, RAD is accompanied by a number of different tools that makes it even friendlier for users.

Today many companies are adopting the incremental release strategy based on RAD. This generally makes them far more effective because testing is done at multiple points during the delivery process instead of at the end of the project (Beynon-Davies P. , 2002). This creates companies that are far more effective and efficient in today's competitive world thus yielding better results and enhanced productivity.

Rapid Application Development is easy to understand in terms of the importance of the performance analysis and predictive tuning in the system. Identifying the process flow initially, especially the limitation of the process, gives an advantage to a company in that they can cut down on the cost of fixes in the later stages of a project and can also save time (Martin, 1991). Over the years RAD has been recognized as an efficient software development process and is considered a valuable strategy.

However, nothing is perfect in this world. Like other project management life cycles RAD contains some inherent flaws that need to be fixed. In order to take complete advantage of RAD by reducing these flaws, an organization must make an investment of both time and money.

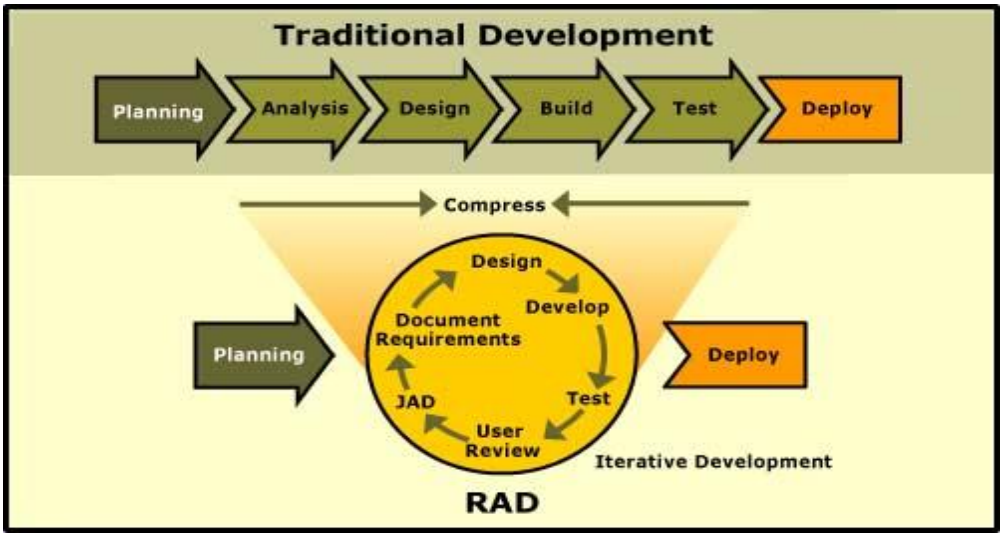


Figure 12.1 Traditional Development Life Cycle vs. RAD

History of Rapid Application Development

Rapid Application Development was used in the mid-1970s to describe the first successful software development process by the New York Telephone Co.'s System Development Center. At that time the director of the company was Dan Gielan. This process was implemented on a number of software development projects and all of them were successful (Janssen, Rapid Application Development (RAD)). After reviewing the success of RAD, Dan discussed the methodology, practice, and benefits in different forums. Subsequently, it became a common name to software developers around the globe.

Later, in 1990, James Martin wrote a book named RAD, Rapid Application Development. In this book he gave a detailed description about the methodologies of RAD (Martin, 1991). However, in this day and age, RAD is used in a much broader and more general way. Now, it encompasses many applications that are being used in the rapid development of software.

Rapid Application Development was basically created in response to the waterfall models and Structured System Analysis and Design Method (SSDM) that were introduced in the 1970s and 1980s. One of the biggest flaws with the previous life cycles was that they simply took too long. By the time a project or product was completed, new requirements arose and the product being released was of no use or inadequate.

So over the years, many developers worked on RAD and tried to make its performance more effective and efficient. Then in the 1980s, at IBM, RAD was effectively

finalized. By 1991 a book was published that detailed RAD's new methodologies and development techniques and proved to be a valuable asset to developers as well as organizations (Report).

How to use RAD?

Most of the RAD processes allow analysts to use practices and computer tools to decrease the time in the analysis, design, and implementation stages. For this purpose they use tools like CASE (Computer Aided Software Engineering), JAD (Joint Application Design), and many other visual programming languages that can simply speed up the whole software development effort. These tools can create the visual appearance of a product as well as generate code based on design specifications.

SDLC phases, tools that generate layouts of software products, and the effective techniques of RAD, together have not only reduced the time to market for software products, but have also increased the quality. RAD methodologies have turned out to be a blessing in disguise for developers today. There are, quite simply, two common methodologies of Rapid Application Development: phased development and prototyping.

Phased Development

The Phased Development process is based on a number of steps. First, it breaks down the overall project into a series of versions that are developed sequentially. During the analysis phase all of the possible components involved are identified and then rated with an importance level (Agarwal, Prasad, Tanniru, & Lynch, 2000). The most important components, as well as the basic requirements, are included in the first version of the product because they will take the least amount of time to complete. The analysis phase is followed by design and implementation.

Work is carried out on versions sequentially. Requirements are implemented on the fly so that the time to market is reduced. While this process may sound quite long and time consuming, in reality it is more effective and efficient compared to traditional methods.

Prototyping

The prototyping methodology performs the analysis, design, and implementation phases all at the same time. All three phases are executed over and over again until the project is completed and the end product is working efficiently. In this case, the users of the

system are an active part of the product delivery process and they are allowed to contribute to the development effort with their judgments and changes that they want to see in the process (Howard, 2002).

Once the initial prototype is completed it is turned over to the end users for testing and feedback. This feedback is analyzed and product redesign is initiated to produce a second prototype. The process continues in a cycle until the users and developers agree on the final product. As before, while this process seems time consuming it has the inherent advantage of constant end user feedback during the course of the project. Historically, this has been shown to ultimately save time by avoiding issues in later stages of the project.

RAD's Four Phases

Rapid Application Development is based on a total of four phases and only after the completion of all phases is a project complete. These phases are shown in the figure below.

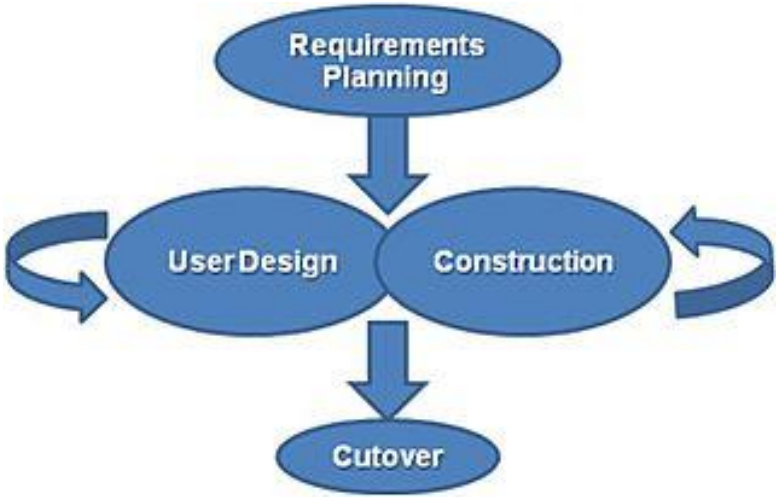


Figure 12.2 The Four Phases of RAD

Requirements Planning Phase

During the Requirements Planning Phase, a thorough review is conducted of the project in an effort to identify success criteria. This is the time that developers, end users, managers, and the rest of the project team discuss the business needs, constraints, scope of the project, and all other requirements (Viceconti, et al., 2007). These discussions continue until the complete team agrees upon the requirements for the effort. Upon agreement of all key issues, the team obtains management authorization to move forward with the project.

User Design Phase

Throughout this phase the end users have the opportunity to collaborate with system analysts and they create models that represent all the inputs, outputs, and processes of the final product. The Rapid Application Development team will typically leverage JAD (Joint Application Development) techniques and CASE tools, so that they can translate the end user's needs into the working models. This phase is all about the continuous interactive process that allows the end users to understand the final product and modify as necessary until eventually the model that best fits their requirements is approved.

Construction Phase

In this phase work is completed on the development of the final product similar to SDLC. However in RAD, the end user still has the chance to participate in the construction and make modifications in flight as necessary. The work included in this phase is mostly application development, integration, and testing.

Cutover Phase

The RAD Cutover Phase is similar to SDLC's implementation phase. It includes data conversion, testing, cutover to the new system (as the name implies), and end user training. With seasoned team members, this entire process is quite efficient. As a result, the new product is released faster, sent to the end user, and is operational much quicker.

Features of RAD

- Projects that use the RAD life cycle generally meet all the requirements of the end user at completion.
- RAD is an extremely effective process for software development.
- RAD usually does not require a major commitment in effort compared to other life cycles.
- RAD produces high quality results.

In short, rapid development, low costs, and high quality are the key features of Rapid Application Development that make it such a huge success.

Success Factors for Rapid Application Development

Rapid development does not mean just any process that reduces delivery time while producing poor quality. There should never be a compromise between the speed of delivery and quality. Rapid development, as the name suggests, means development that takes place in a relatively short period of time compared to traditional processes (Singh, 2012).

Many believe that products that are delivered quickly are not reliable and are of poor quality. They feel products that took longer to deliver were better planned and subsequently of higher quality and more effective upon release. However, as stated above, history has shown that while RAD reduces delivery time there are generally no adverse effects on the quality of the work.

A number of factors play an integral role in the success of Rapid Application Development. These factors not only influence the quality of the final product delivered, but also the speed at which it is delivered (Woods, 2010). Through the embracing of these success factors RAD becomes and effective project management life cycle. Some of those factors integral to RAD are documented below.

- There must be adequate requirements in order to conduct workshops or focus groups. Only by having all the essential elements on hand can effective communication take place.
- All the components of the final product must be tested and prototyped.
- Team members should constantly look for opportunities to reuse components of the final product. With the use of CASE, proven templates, components, and processes can be reused as identified.
- Integrated use of the CASE toolset may help to enforce technical reliability in modeling and designing of the final product in a way that has not been seen before as well as produce error free code with the help of a fully validated design.
- A realistic, but fast paced schedule helps to ensure suggested design improvements are incorporated into the next product release rather than creating scope creep and slowing down the project.

- As mentioned before, this is another project management life cycle that leverages the engagement of the end user throughout the whole project. This is a proven practice to help in change control at the point where a project begins to deviate from what the end user has conceptualized and expects to have at the end of the effort.

To expand on the bullet above, the success of Rapid Application Development substantially relies upon the early, effective, and continued involvement of end users in the delivery process. This involvement is so critical it can solely make or break a project. Effective involvement of end users is generally considered the single most important key factor in the timely discovery of errors. It has been proven that the earlier a defect is found the less time and expense it will require to be corrected.

So, quite simply, the later you find an error, the more costly it will be. Complete understanding of this concept by all team members will help drive keen observation at each and every step to identify issues and increase project performance. While it sounds over simplified, the team just needs to ensure that when they move to the next step they are certain that the previous step was completed with no issues.

Evaluation of RAD

By now it should be obvious that RAD has proven to be extremely versatile and valuable for project management especially within the software realm. However, as alluded to at the beginning of this chapter, even RAD has its pros and cons. It is up to the project team and its stakeholders to ascertain whether the cons are worth all the benefits that the project life cycle can deliver (Beynon-Davies P. , 2002). Some might find it helpful while others might not have a very positive opinion of the methodology. We discuss the advantages and disadvantages in greater detail later in this chapter to help guide your decision.

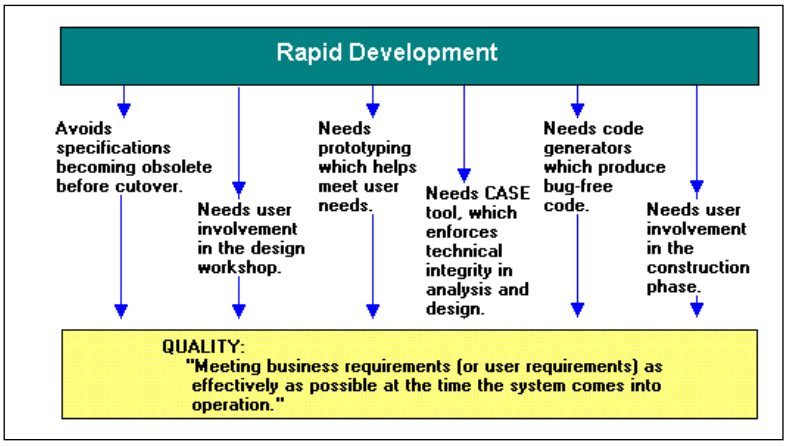


Figure 12.3 RAD Requirements

The Rapid Development Threat

Over a period of time many companies have become increasingly dependent on Rapid Application Development. There is, however, a potential business threat that comes with this dependency. Huge, multi-national companies are often tempted to use RAD techniques in order to build standalone products for solving a specific problem in isolation to the rest of the company. If such products manage to meet the needs of the users, they subsequently become institutionalized.

If a large company ends up producing a great number of such isolated systems to solve particular problems the outcome will be a large, unorganized accumulation of products that do not work together (Beynon-Davies, Carne, Mackay, & Tudhope, Volume 8, Number 3, 1 September 1999). In the real world, or at least an ideal world, most business applications are closely interlinked to other applications and share databases with each other making a common infrastructure essential. It's always important to remember to never lose sight of the big picture. Thus, working in a vacuum must be avoided.

Additionally, with the passage of time most computer networks grow either through organic growth or even mergers. As they grow they tend to become even more complicated and such networks are extremely difficult to change unless they have grown in alignment with the constantly changing greater picture. Therefore, the solution to any problem is to develop a complete set of requirements for every project with the inclusion of all stakeholders. These stakeholders must be people from all areas where communication takes place or could take place in the future. This concept has often been referred to as "Information Engineering" and it is important to keep in mind that computer networks are

only used here as an example because many projects have interdependencies such as an airplane's airframe and engines or multi-departmental processes within a corporation.

Core Elements of RAD

Quite simply, prototyping, incremental development, time boxing, the team members, the management approach, and the RAD tools are some of the core elements of the Rapid Application Development methodology that makes it unique from other project management life cycles. These elements and more are discussed in further detail below.

Prototyping

Prototyping is a key aspect of the RAD methodology. In this way products are created according to the requirements of the end user. The prototype is often a lighter version of the finished product that is made in just a few days. It is basically used as a communication tool for the client and project team to validate their understanding of the project requirements.

Incremental Development

Incremental development simply means the release of small functional versions of the final product. Every version is reviewed with the client for approval and more requirements are fulfilled in the next release. This process continues over and over again until all of the final functionality is delivered and meets the requirements of the end user. The recommended time frame for completion of incremental releases is a maximum of three weeks.

Time Boxing

Time boxing is simply a way to organize work and create focus for the team. It is considered by some to be one of the most important aspects of the Rapid Application Development life cycle and should be strictly followed. If this process is not followed, the project is at risk of falling behind schedule.

Team Members

In Rapid Application Development teams should be small groups of the best people. They should consist of experienced, versatile individuals that can multi-task. All team members must be motivated towards completion of their tasks by all required deadlines. As the end user plays a vital role in RAD, they must be available for group discussions with the team members before and after each incremental release. This will ensure alignment of the

team with the end user's requirements. For the sake of efficiency during the project, the team members selected must be familiar with Rapid Application Development and its tools.

Management Approach

An active and involved management team also plays a critical role in projects that leverage RAD. These people are the escalation points for the project and work to resolve misunderstandings between team members as well as customers. Additionally, they assist with project discrepancies and drive deadlines. Furthermore, the management team must be steadfast and consistent with what they need from the project team. Finally, the management team is the group that often establishes timetables for the project, selects the team members, provides motivation, and manages the inherent politics that comes with any project.

RAD Tools

Rapid Application Development, as the name implies, was basically designed to take advantage of the latest tools as they are introduced to the market. Unfortunately, not all tools support RAD completely. As is the case with many tools, they are often only associated with RAD for marketing purposes (Agarwal, Prasad, Tanniru, & Lynch, 2000). Some of the products that are normally associated with Rapid Application Development tools are listed below.

Data Integration

When it comes to Rapid Integration Tools, especially for Rapid Application Development, data integration is always a factor. Often though the systems on the market are meant to be versatile for all software and their primary functions may not make them perfect for a RAD environment. They can be more suitable for SCRUM and waterfall project management life cycles.

Development Environments

In October 2004 an article written by ZD Net was published in which they compared the tools that were currently being used for Rapid Application Development. The name of the article was quite simply Five IDE'S Tested. The article was useful in convincingly comparing all the tools available though they were all just development environments for RAD. While just using the Rapid Application Development life cycle for development purposes, these tools still played a vital role. They would genuinely make the development process quicker as

compared to other tools. However, none of them seemed to be as supportive to RAD as Microsoft Project when it came to the opinions of some.

Requirement Gathering Tool

A project is generally not terminated successfully until a requirements work search is completed and all data is collated. While this is yet another life cycle that strives to reduce paperwork, everything must be properly maintained instead of collecting the data in random documents. To accomplish this many tools have been created but one of the most popular is known as UML (Unified Modeling Language) and it is being used in many software development houses. UML supports RAD, so all data regarding a Rapid Application Development project can easily be saved and secured with this tool. Additionally, UML is supported by other tools like Microsoft Visio and IBM's Rational Rose.

Data Modeling Tools

Like requirement gathering, data modeling is also a vital part of the RAD life cycle process. However, developing a database manually with the use of SQL or an integrated development environment (IDE) can be counter to the RAD process. This is because it is very possible that these tools and the process involved may be too time intensive and ultimately slow down the RAD process. However, not using a data modeling tool may also affect the Rapid Application Development process adversely.

That said, as mentioned above, often tools that support UML and requirement gathering have data modeling tools incorporated with them or companion tools. Microsoft Visio is one such tool that can be used for this purpose. It not only supports the Rapid Application Development process, but also has most of the tools needed for RAD support like requirement gathering tools, UML, data modeling tools, and data design tools. For most projects, Microsoft Visio is everything in one box. You just have to open it and start designing the applications needed by your customers.

Code Generation Tools

RAD was designed so that developers could take advantage of CASE. CASE is a tool that incorporates requirement gathering, data modeling, and allows for subsequent automated code generation. In code generation a developer can use the inputs from CASE and transform

them into the source code with automated code generation tools. This avoids having to write code in traditional fashion following processes that are time consuming.

The rapid growth in technology has allowed many companies to develop numerous code generation tools. These tools can now allow for easy creation of source code from CASE inputs. Of course this evolution continues to shorten project durations and allows for faster product releases.

Advantages of RAD

There are a number of advantages of RAD and they are listed below.

- Rapid Application Development was designed to deliver products faster with minimum cost and high quality.
- Final products are usually easier to port.
- Development of software is conducted at a higher level because RAD tools tend to operate more efficiently at that level.
- RAD offers the opportunity for end users to have early visibility of the end product through prototyping.
- RAD is a project management life cycle that favors flexibility and gives developers the empowerment to redesign as needed and reduces manual coding with the use of wizards and code generators.
- The reuse of code makes it easier to deliver products faster.
- Rapid Application Development leverages CASE tools to shorten development cycles and reduces defects.

Disadvantages of RAD

Like other project life cycles, Rapid Application Development is not the Holy Grail for all projects. Even this process has room for improvement. Some of the disadvantages of RAD are addressed below.

- The cost of tools can be extensive but if an organization is committed to RAD this can end up an investment that pays for itself many times over in the long run.

- It is often harder to measure progress on RAD projects because classic tools and processes are not used.
- RAD can seem, at times, less efficient than more traditional project management life cycles.
- The RAD project management life cycle can easily stray from precise processes and results due to the lack of formal methods.
- While using RAD it is possible to inadvertently return to the uncontrolled practices of the early years of software development.
- Requirements are likely to change frequently due to the interests, choices, and needs of end users varying from person to person.

Conclusion

Rapid Application Development is without a doubt a viable, flexible, project management life cycle. However, like anything else in this world there is good and bad to everything. All RAD needs to be a successful life cycle is the right combination of techniques, tools, team members and management. Various factors do need to be considered before making the decision to use RAD for a project though, including the architectures and infrastructures involved. Rapid Application Development will surely continue to evolve as a project life cycle and become even more effective.

There should be no uncertainty that RAD was created for completing a project in the fastest time possible in order to deliver the end product efficiently and effectively. James Martin planned for it to be the fastest approach possible for application development (Coleman & Verbruggen, 1998). Every phase in Rapid Application Development is crystal clear. Furthermore, after each and every phase the product is reviewed with the client for feedback to incorporate in subsequent updates.

This process repeats until the final product is complete and meets all of the end user's requirements. RAD not only reduces project duration but also serves to reduce costs. While RAD was created in the early 1980's these results are, of course, desirable enough that the project life cycle is still used. Many companies believe it is simply the best approach for project delivery and cost reduction. Future research into improved techniques and

implementation factors will further reduce the inherent faults with Rapid Application Development.

Chapter 13

Unicycle

Unicycle is generally considered one of the earliest project management life cycles and it is still used today. According to many researchers, the exact birth date of this life cycle is unknown (Allan, 1997). One of the key aspects of Unicycle is that the project manager is given the ultimate authority throughout the project with regard to all decisions. However, the life cycle failed to find wide scale acceptance and thus was not used broadly for development and project management.

What is Unicycle?

As mentioned above, Unicycle is one of the earliest project management life cycles and it is known for its formal structure. It is comparatively simple to other life cycles because it is easy to trace the steps in the process. However, like many project management life cycles, Unicycle demands efficient communication throughout the project management process and phases are tightly joined. Additionally, instead of an incremental approach, this life cycle follows a phased approach.

Unicycle is a highly structured project management life cycle (Azam, 2010). As with many Agile life cycles, the end users' involvement in every phase is extremely important. This is assured through continuous communication throughout the project. Essentially, business and technical tasks make up the phases of the Unicycle project management process.

History of Unicycle

Oddly, there is no definitive information available on the history of the Unicycle project management life cycle. It is a bit of enigma in that nobody has documented when it started or how it evolved over time.

Feasibility Report

As part of the process, a feasibility report is developed after thorough study of the proposed project plan to ensure that the project makes good use of the resources that are assigned. The project manager also needs to ensure that the feasibility study conducted is up-to-date and reflects the current status of the project environment (Charvat, 2003). If a team

considers the use of this project life cycle it is always a good idea to start with a feasibility report to determine whether or not the chances of success rate are realistic or not.

Requirements

Of course, no project can be successful if it does not deliver the intended requirements. Thus it is important to define the expectations for the final outcome of the project. This of course means that a detailed analysis must be conducted to identify all requirements.

Design Documents

Once the feasibility report and requirements document have been completed the next step is to document the high level system design to guide development. When the design document is ready the work can be divided into work packages and scheduled.

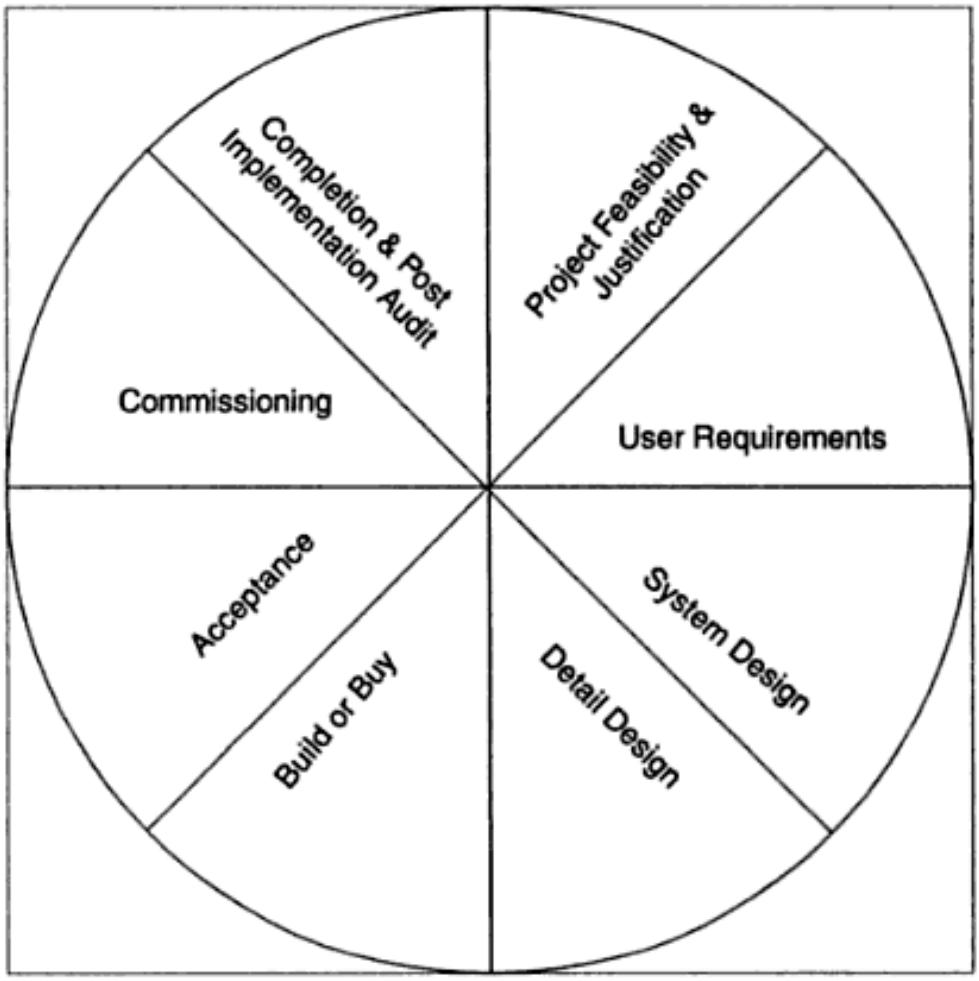


Figure 13.1 Unicycle Process

Work Breakdown Structure

Having completed the high level design, the next step is to create work packages and schedule activities. This is generally documented via a work breakdown structure (WBS). The completion of the WBS represents the finalization of the design phase.

Testing and Validation

Acceptance of the final product is vitally important for the success of any project. As such, after the development work has been completed it is necessary to perform testing and validation to ensure that everything is per the requirements. Generally, both unit and system tests are performed on any final product of the project.

Acceptance and Implementation

Once all project work is complete, formal approval is sought from all stakeholders. If the final product of the project satisfies all requirements, this is typically an easy task. That said, final acceptance and phase acceptance are two different things. If, however, phase acceptance has gone smoothly during the project, final acceptance should go equally well (Azam, 2010).

Unicycle as a Heavy Methodology

Unicycle is considered a heavy methodology or life cycle because of such aspects as bureaucracy, obsolescence, slower design and development process, prediction of technical details, longer project durations, and the general lack of accommodation for change (Charvat, 2003).

For all heavy project life cycles like Unicycle it is extremely important to spend time getting comprehensive requirements. Scope control is rigid and changes later in the project are difficult (Charvat, 2003). This process for documentation is far different from Agile and an increased amount of documentation is required for success because all team members need to know what they are supposed to do. Communication with a heavy life cycle is through documentation.

With these project management life cycles it is important to have support of top management and the project team so that frustrations and complexities can be avoided. Thus

with heavy life cycles it is important to have greater focus on project details so that success is achieved.

Advantages of Unicycle

There are a number of advantages of Unicycle and they are listed below.

- If a project is completed using Unicycle generally the risk level is low.
- This is a project management life cycle that offers ease of implementation and an increased reliability for the project team and project manager (Charvat, 2003).
- When correctly executed, the project becomes much easier to manage.
- This life cycle strongly embraces the triple constraints of project management.
- The project manager can adapt the life cycle according to the needs of the particular project.
- The Unicycle project management life cycle can accommodate all project sizes from small to medium and even large projects (Charvat, 2003).
- Unicycle tends to focus on the technical aspects of a project (Azam, 2010).
- Unicycle is generally easy to manage as well as reliable.
- Formal phase exits increase communication during the project.

Disadvantages of Unicycle

There are a number of disadvantages of Unicycle and they are listed below.

- Changes cannot be easily implemented.
- Extremely little information is available about Unicycle and thus there is learning and implementation issues with it.

Code and Fix

Code and Fix strays a bit from main stream project management life cycles. As the name implies it is a life cycle used for software development. It works on the principle that a project can be completed successfully without having been properly planned. It is believed that all software developers have to do is start coding and any problems that appear along the way (as they most definitely will) will be fixed on the fly. Many software development life cycles have evolved from the Code and Fix approach.

Although this may seem like a naïve approach it has been shown to have some success. However, it does not yield a very good success rate although it remains a common approach with some developers (Giunchiglia & Tomasi, 2005). The wide use of this approach is not due to its success though but rather the simplicity it provides. The Code and Fix approach is also known by the name Build and Fix (Stein, 2004). Other informal names for the process include “hacking,” “immature development,” “no process at all,” and “hack and slash” among others (Ambler S.).

While a project may be completed sooner, all the testing and debugging that is usually required with this completely unstructured approach costs a lot more time and money than other project management life cycles (Thibaut, 2013). This is an approach that basically disregards all standards of software engineering.

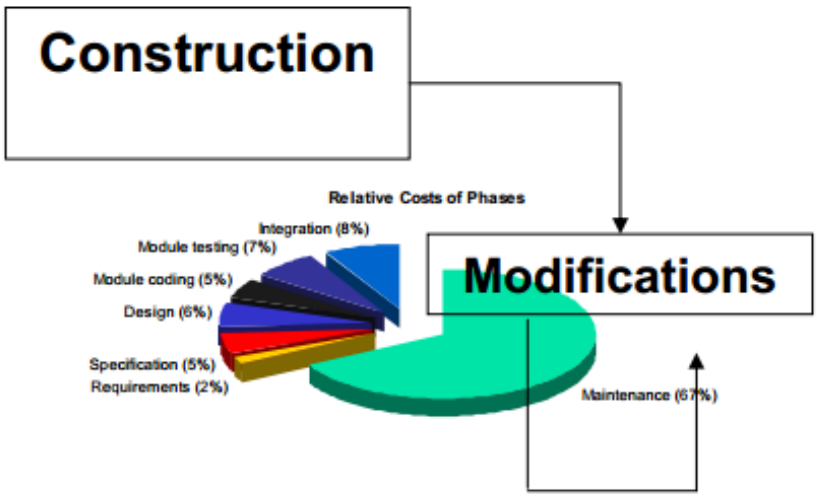


Figure 13.2 Division of Resources within the Phases of Code and Fix

What is Code and Fix?

Code and Fix is a project management life cycle that supports a freestyle method to design by eliminating formal processes and still striving to achieve results. Formal stages like analysis, quality control, requirements definition, documentation, and design are either skipped altogether or they are merged within the Code and Fix process (Takang & Grubb, 2003). If no formal processes are being used on a project then most likely it is a Code and Fix effort. Quite simply, while using other project management life cycles the team must be vigilant in adhering to the life cycle or risk the project degenerating into a Code and Fix model.

As stated, there is basically no adherence to coding standards, and every programmer is welcome to follow any approach that they like. As a result, the code developed using this method has often been referred to as spaghetti code (Boehm B. W., 1988). Team members are allowed to reuse any code that they need and find ways around any formalities to achieve their end in the easiest manner possible (Thibaut, 2013).

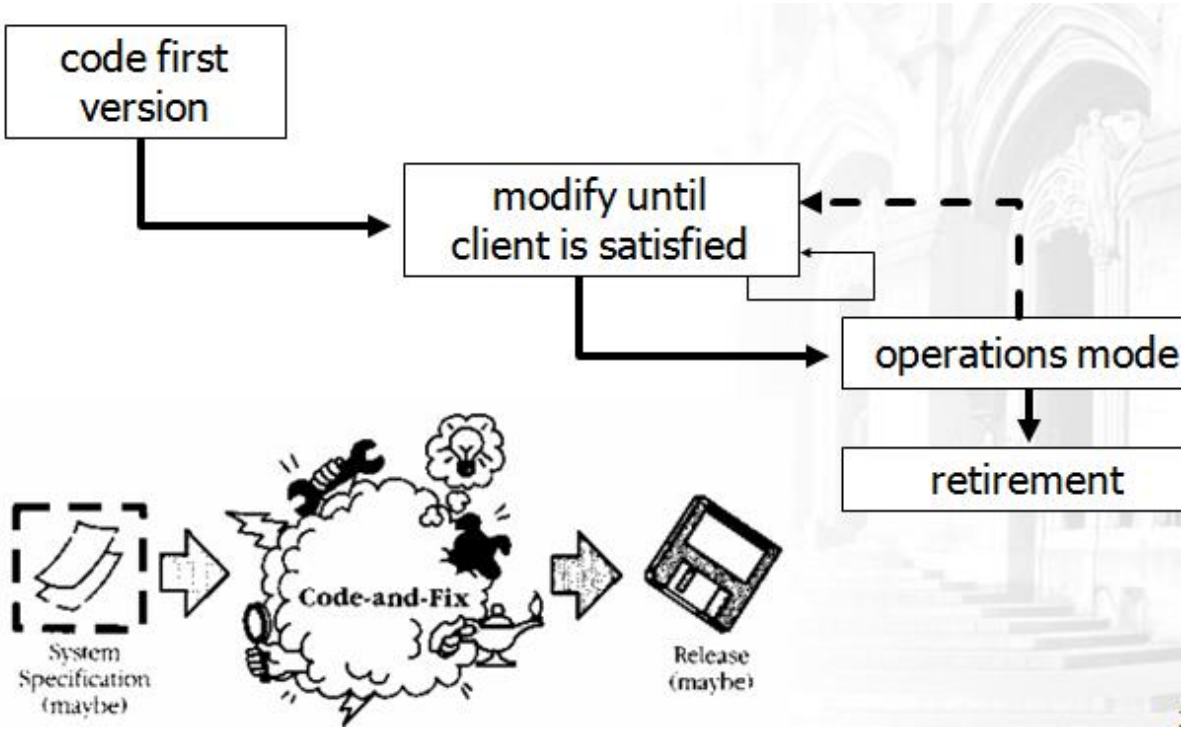


Figure 13.3 Figure Shows a Code and Fix Model

Instead of frequent product versions being released to the market or sent for approval to management, Code and Fix relies on a single beta test for product approval. Every programmer is responsible for their piece of code only (Thibaut, 2013). No one can interfere

in another person's work and there is generally no sense of team work. Additionally, testing is complete on the whole product at one time instead of on units or modules. Code and Fix is characterized by little or no documentation. Everything is completed on the fly.

History of Code and Fix

Not surprisingly, there is no definite information available on the origin of the Code and Fix life cycle. The general consensus is it started evolving somewhere in 1950s. Later, it was given the informal name of Code and Go – or – Cut and Run (Paul, 1997-1999). Today it is also known as the Cowboy approach because of individual developers who refuse to abide by somebody else's processes. It almost goes without saying that early software development used a Code and Fix approach (Boehm B. , May 1988).

Code and Fix became widely prevalent in the 1970s. Later, the Waterfall project management life cycle was designed to bring formality to software projects and get away from the problems of Code and Fix (Admin, 2013). Collectively, along with the Waterfall and the Spiral project management life cycles these methodologies are considered traditional process models (Stafford, 2003).

Some have said that this software development approach was born from hardware engineers in the early days of the automobile industry (Jayaswal & Patton, 2006). Early cars were developed and not tested before handing over to the customer. The consumer then tested the car and if any problems were reported the manufacturer readily fixed them. It is thought that initial software development carried this model forward.

When to Use it?

It is not advisable to use Code and Fix on large projects. Additionally, when requirements are varied and of paramount importance then this method would not yield a successful result. This is ad-hoc project management and will not be able to deliver on wide-ranging requirements. However, for teams still honing their skills with room to fail, this can be a useful process because they can resolve issues and learn from their mistakes as many times as required without needing to worry about process.

Code and Fix can also be useful when requirements are undefined or ambiguous. The team can kick off the project and keep changing or improving the final product until the end user is satisfied. The upside to little or absolutely no planning is that it reduces management

overhead on the project. Additionally, it is an effective life cycle for projects under 300 lines of code. However, if development will extend beyond a day or two it is not advisable to use Code and Fix.

Code and Fix can also be used for small personal projects. A single person developing by himself has far more flexibility and can work without a plan. Most of the time, even if a rough plan is created it is usually, promptly disposed of (Ambler S.). Any failures that arise later can be fixed in the testing stage. Additionally, as mentioned before this also allows novice developers to work on small projects to polish their skills without constraints or formal processes.

Code and Fix is also suitable for disposable projects or demo products when the development team does not need to follow a rigid life cycle and go through planning and documentation for a low level effort. The purpose of such short lived projects is usually to prototype a product and Code and Fix will serve this purpose. Essentially, whenever, a project team does not adhere to any specific standards and structured approach, it is said to work on the foundations of this project management life cycle.

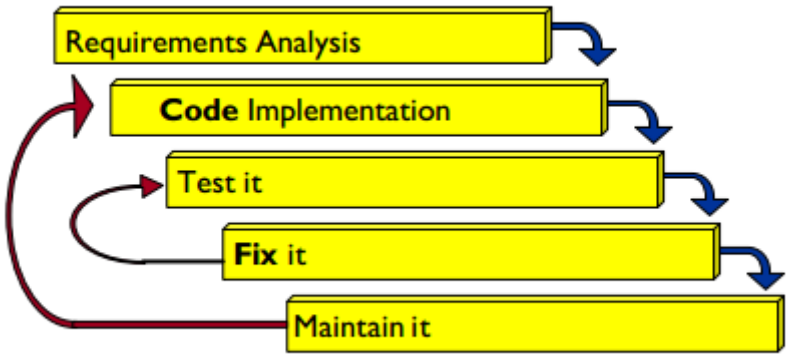


Figure 13.4 Code and Fix Process

Maintenance Requirement

As no specific design is created before the work begins, the Code and Fix model requires loads of maintenance. No doubt it is the easiest approach for starting work, however, when it comes to completing a project debug and updates can become nightmarish. Owing to all of the resources required as well as the time and money spent during the maintenance and

debugging phase Code and Fix becomes one of the most costly approaches to project management.

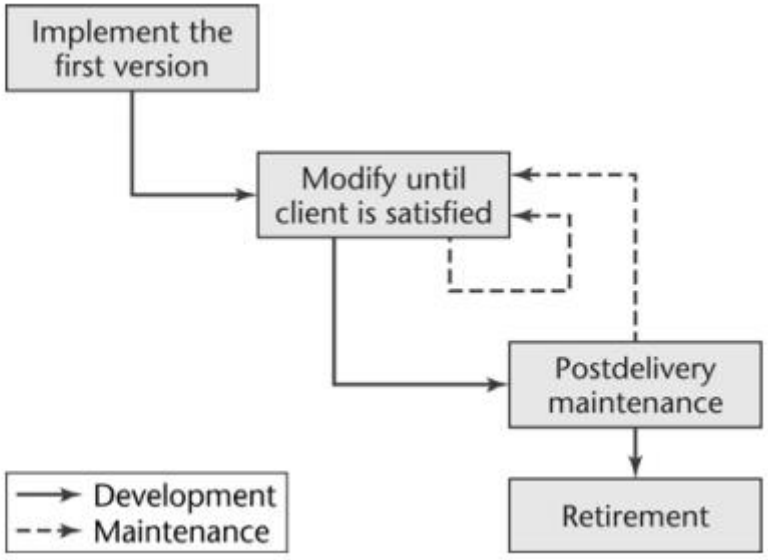


Figure 13.5 Maintenance Requirements in Code and Fix

Inevitably the initial working product that is produced using Code and Fix will be inundated with errors since there was no predefined design or coding specification. The project team first codes and then considers the requirements before fixing. Historically, problems identified late in a project have been more expensive to resolve and Code and Fix is no different.

At times, laymen confuse Agile with Code and Fix owing to certain similarities between the two. However, upon closer inspection and observation of project teams the differences in the life cycles becomes apparent.

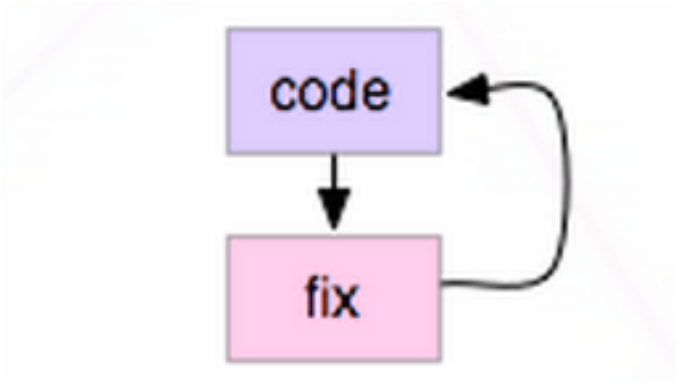


Figure 13.6 Simplest Possible Illustration of Code and Fix

Advantages of Code and Fix

There are a number of advantages of Code and Fix and they are listed below.

- Time and resources normally spent on analysis and planning are saved and the development team can begin work immediately (Giunchiglia & Tomasi, 2005).
- There is no rigidity of structured processes and thus no visible transition from one phase to another.
- No formal acceptance is required at every step of the life cycle.
- It is the easiest and simplest project management life cycle.
- Code and Fix can produce results faster than other project management life cycles.
- Since code development starts quickly results are immediately visible.
- Since they are no procedural standards work starts quickly.
- The developers can go back to any development stage at any time until all the errors are fixed and the customer requirements are satisfied.
- This project management life cycle tends to suit companies with fewer employees. Smaller companies generally cannot afford to allocate many human resources to one project, so it is better for them to let a few resources proceed with a Code and Fix approach.
- Code and Fix is also suitable for instances when the stakeholders involved are few in number allowing easier management of requirements.
- With this project management life cycle, testing is completed only once, which reduces the complexities inherent in formal testing procedures.
- Young developers who need to hone their skills can begin with a Code and Fix approach and later transition to other project management life cycles.
- Very little to no experience is required to implement this project management life cycle.

Disadvantages of Code and Fix

There are a number of disadvantages of Code and Fix and they are listed below.

- Projects using Code and Fix can find it to be rather difficult when handling modifications or changes.
- There are generally no intermediate prototypes so the end user has to wait until the final product is released in order to evaluate it (Giunchiglia & Tomasi, 2005).
- As no documentation is produced for any projects completed using a Code and Fix approach, if anyone needs to retrieve information about the project later it is impossible.
- Quality of the final product is often an issue since code is created without the use of any formal standards and procedures.
- Products completed with the use of Code and Fix tend to be poorly maintained (Giunchiglia & Tomasi, 2005). Additionally, as the amount of code increases, confusion about the overall product increases.
- If the project manager or the delivery team decides to switch to another methodology midstream in the project it is not an easy task.
- The delivery team or organization using Code and Fix can become so comfortable with a no formality process that they seldom if ever want to adopt a more rigid project management life cycle.
- It is nearly impossible to give accurate estimates for time and cost with the use of this project management life cycle.
- Changes late in the project will consume significant resources.
- While the easiest project management life cycle to implement it often tends to be the most costly project management life cycle.
- Extended testing times can result in cost overruns.
- Extensive time spent testing and debugging can cause delivery team morale and motivation to drop considerably.

- Late project changes can also significantly setback a project and sometimes the project budget or schedule is exhausted and development is stopped mid-way through the effort.
- If the final product delivered is substantially different from the end user requirements, it is possible that the team may need to scrap all work completed and start over from the beginning.

Scaled Agile Framework (SAFe)

Scaled Agile Framework is an interactive knowledge base for the implementation of Agile practices at the enterprise level.

History of SAFe

This Agile Software Development methodology was brought forward by Scaled Agile, Inc. Version 1.0 that was released in 2011 (Scaled Agile Framework, 2016). The best thing about SAFe is that an organization can mold it to its needs. Dean Leffingwell was the pioneer who explained and propagated SAFe not only in his books but also in his blogs (Scaled Agile Framework, 2016). 2016 has saw the release of the latest version of SAFe in the form of SAFe 4.0 for Lean Software and Systems Engineering.

Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it."

Through this work we have obtained the values below.

- Individuals and interactions over processes and tools.
- Working software instead of comprehensive documentation.
- Customer collaboration as opposed to contract negotiation.
- Responding to change instead of following a plan.

This ultimately means that continuous improvement is part of Agile along with people, products, collaborations, delivery, customer collaboration, and change. The framework that is called Agile must fulfill all of these attributes (Saddington, 2014).

Purpose

The goals of Scaled Agile Framework (SAFe) are listed below.

- Increase the productivity, enhance the competitiveness, and increase the quality of software delivered for the worldwide software industry.
- Provide the business benefits of Agile development to all software teams.

- Increase motivation, enjoyment, empowerment, and humanity to software developers everywhere.

SAFe Position Statement

This framework is for practitioners, developers, executives, managers, coaches, consultants, and Agile team members who are involved in planning and implementing Agile software development philosophies, practices, and principles on an enterprise scale. Scaled Agile Framework is a publicly available and proven framework for implementing practices on an enterprise scale that is presented in an interactive web format (The Horror Of The Scaled Agile Framework, 2012). Unlike other proprietary methods that are promoted by consulting firms, SAFe is a holistic, summarized and integrated framework that is available to everyone everywhere and is readily accessible on a public-facing website along with other resources.

The Scaled Agile Framework Interface is a "Big Picture" graphic that highlights the roles of individuals, teams, activities, and artifacts essential to scale Agile from the team level to an enterprise level.

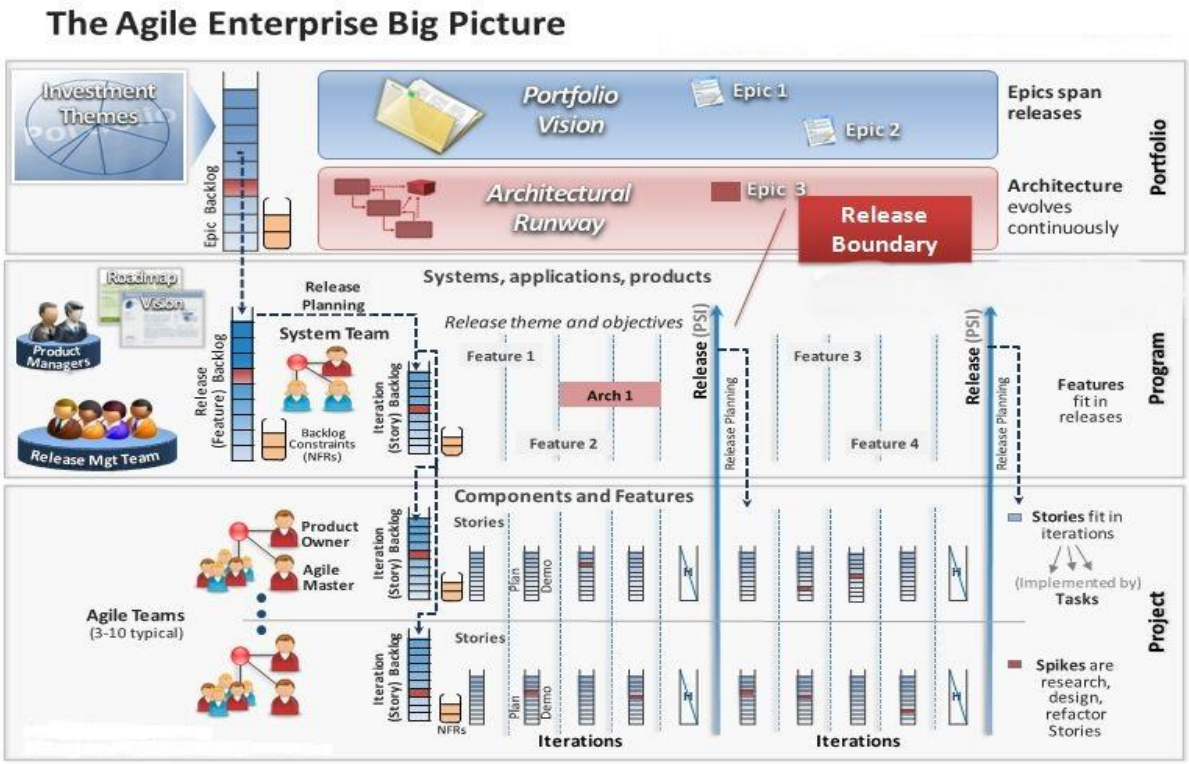


Figure 13.7 The Agile Enterprise Big Picture

The big picture in Figure13.7 elaborates three levels of scale that are listed below.

- Team.
- Program.
- Portfolio.

It shows that product development is divided into these three levels mentioned above.

Team Level

Several tasks take place at the team level. At this level Agile teams comprised of 7 to 9 members each are defined and built. Next, these teams test users' stories in an arrangement of iterations and releases. There are only a few such teams present in small enterprise environments, but in large enterprises, pods or groups of Agile teams collaborate or work together in order to build larger functionality into complete products, architectural components, features, subsystems and much more. Teams need to be assigned to a product owner in order for him to manage the backlog of users' stories.

Figure13.8 Members of the Team Level

Different teams have different tasks and there are even some Agile teams on the frontlines whose task is to create and implement test code. The use of teams is a fundamental part of Agile because they deliver value through collaboration (Elssamadisy, 2013). The roles of a team consist of an Agile master, product owner, and a small number of developers, dedicated testers, and some test automation experts. The team may also include a technical lead. On daily tasks, the teams are supported by architects, external resources, specialists in documentation and database management, source code management specialists, internal IT personnel, and others to make sure the core team is completely successful in planning, developing, testing, and delivery activities.

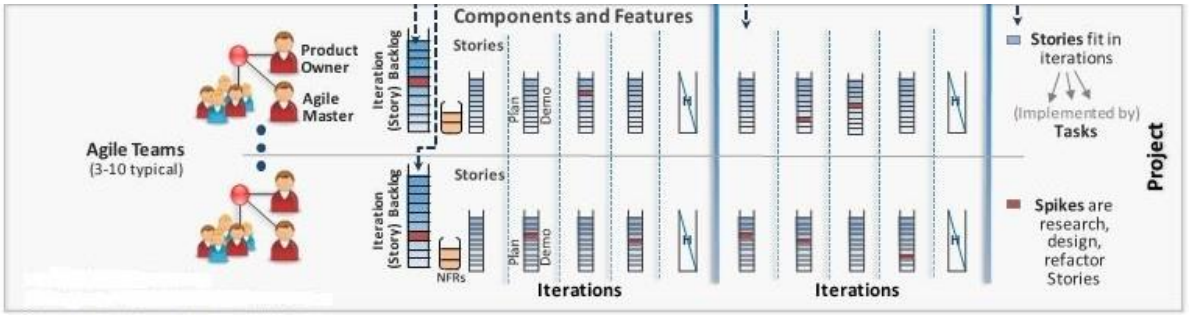


Figure 13.9 Team Level of Big Picture

Roles and Components of an Agile Team

While roles of an Agile team have been included in other areas of this book, we will touch on them again here for consistency and ease of reference.

Product Owner

The Product Owner is a principal Agile role that is distinctively defined. Some of this role's main tasks include managing backlogs and determining and prioritizing user requirements. This person can greatly simplify the team's work with organizing all activities around a prioritized backlog.

However, the responsibility of the product owner doesn't end with a prioritized backlog. In accordance with the Agile manifesto, in any given project the business people and delivery team should work together daily throughout the project. This means that the product owner ideally is co-located and tightly integrated with all the teams' activities.

Agile Master

The Agile Master is yet another key role of any team. They are an Agile process expert. Basically, this person is the team-based leadership proxy who helps the team in its transition to and execution of Agile. This role is critical in maximization of performance of the team.

If a team does not assign a dedicated and experienced Agile Master, then this leadership role would typically fall on a team lead or other internal or external coach. As with any developed skill, aspiring leaders have an opportunity here. By representing themselves as Agile experts they may be given the chance to deliver value in this role (The Horror Of The Scaled Agile Framework, 2012).

Developers and Testers

The rest of most teams are comprised of testers and developers. Testers include, of course, the team members who write and test the code. Since we are talking about Agile teams, the team size is limited typically to about three or four developers and about one or two testers. The team is often co-located so that they can physically work together on a daily basis to deliver the project as planned.

Iterations

With Agile, development is fast and incremental. It is also completed during periods of production known as time-boxed events. In larger enterprises and projects, teams typically decide in advance the standard time frame for an incremental release. Additionally, they will also set the start and end boundaries for that release.

The new incremental release always builds on the previous release and the end user's input. This is the trusted premise of the process. It is basically build, review, and build again until final test and acceptance. This ultimately is the plan and review should be conducted with all stakeholders for the best results (Saddington, 2014).

It cannot be stated enough that incremental releases are a critically important factor of the Agile process. For this reason they are known to some as the "Heartbeat of Agility." Incremental releases keep the teams focused only on developing or adding new functionality in the short time-boxes they are given thus driving efficiency.

It takes a series of incremental releases to aggregate system-wide functionality to the end user. This can be an arbitrary pattern and there is no hard and fast rule for how many times a team creates incremental releases to achieve a PSI (Potentially shippable increment). Many teams consider four to five incremental releases to be the norm thereby creating a potential rhythm for a PSI of about every 90 days. This is a natural cadence that corresponds to external release frequency for customers and it provides a quarterly planning rhythm for the enterprise itself.

Team Backlog and User Stories

User stories are traditionally referred to as software requirements. These generally seem to have been developed originally within the constructs of XP, but now, Agile user stories are, in general, typically taught in Scrum, XP, and other Agile project management

life cycles. In short, users' stories are the primary instrument that identifies the customer's requirements.

Team backlog is typically known as product or project backlog. It consists of all users' stories the team has identified for processing, but has not processed yet. Every team has its own backlog that is prioritized and maintained by the product owner. There may be defects, infrastructure work, changes, and more included in the backlog for a product.

Tasks

Teams will typically decompose users' stories into small tasks for execution of the activities involved in delivering requirements. This breakdown approach helps organize the work at hand. All of these tasks must be accomplished by individual members of a team for the completion of requirements.

Each incremental release must be focused on the story level. This keeps teams directed toward delivering on value instead of just individual tasks. Tasks are merely the work packages of a work breakdown structure that teams can utilize to facilitate coordinating and assigning individual responsibilities to assure the completion of requirements and thus the incremental release (leffingwell, 2010).

Program Level

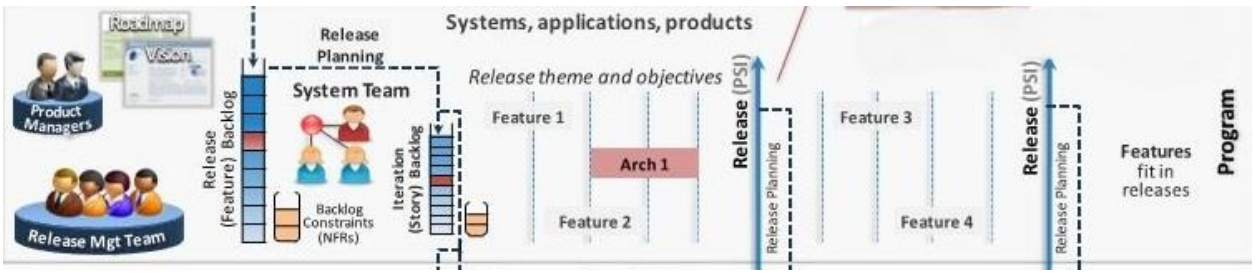


Figure 13.10 Program Level of Big Picture

At the program level is where we find added organizational constructs, processes, roles and requirements that are suited for building large scale systems, products, applications, and product families (Pinna, 2013).

Potentially Shippable Increments and Releases

To produce a shippable incremental release of the final product is the goal of all iterations, but sometimes teams find that it is not appropriate to ship an incremental release at

every iteration-boundary. There are some legitimate reasons for not shipping. We detail some of these below.

- Potential interference with a customer's service agreements and licensing.
- Potential business disruption and the overhead of installation.
- Potential for disrupting the customer's existing operations.

For these reasons most projects aggregate a series of incremental releases versus shipping of a release at each iteration-boundary.

Vision, Features, and Program Backlog

In an enterprise the primary responsibility for maintaining the vision of the product is the function of product management. Similar to a teams' backlog that primarily consists of users' stories, the program backlog contains a set of prioritized and desired features that have not been implemented yet. The program backlog may or may not contain estimations for implementation of the features.

Release Planning

According to emerging Agile practices each time-box for an incremental release has a release planning session that is used by the enterprise to align the business to business objectives for the release. The input to the release planning session is simply vision along with objectives and desired features for the release (Implementing Scaled Agile Framework (SAFe) with VersionOne).

Roadmap

A roadmap basically consists of planned dates of incremental releases. Each of the releases consists of a set of objectives and a prioritized list of features.

Product Management

With Agile product management, regardless of the title of the person filling the role, the duties of the product manager should include the items listed below.

- Own the vision.
- Own the program backlog.

- Manage release content.
- Maintain the product roadmap.
- Build an effective product owner team.

(leffingwell, 2010)

Portfolio Level

Figure13.11 Portfolio Level of Big Picture

Individuals, organizations, and teams that are dedicated to managing enterprise investments according to the business strategy are included in the functions of portfolio management.

Investment Themes

Investment themes establish the investment objectives for a business unit. The vision, as well as new epics, for all programs is derived from these themes. These decisions are part of the responsibilities of the portfolio managers.

Portfolio Backlog and Epics

Epics are simply large user stories. They are development initiatives that are prioritized, maintained, and estimated. These initiatives are intended to deliver value on an investment theme in the portfolio backlog.

Architectural Runway

Architectural runway is the infrastructure to deliver features on current and near term solutions documented in the roadmap without excessive refactoring. System architecture has always been considered a first class citizen of the Big Picture and for Agile enterprise it is considered a routine portfolio investment (Leffingwell, 2014).

Advantages of SAFe

There are a number of advantages of SAFe and they are listed below.

- It has marketing benefits. Management has to be on board with SAFe.
- Scaled Agile Framework is often easier to sell to customers as a project approach.
- It is possible to start an effort where proof of Value can be most easily seen.
- SAFe focuses on Lean thinking and understanding work flow.
- Costs of delays are reduced because of increased efficiency.
- SAFe can be effective in being first to market with a product.

Disadvantages of SAFe

There are a number of disadvantages of SAFe and they are listed below.

- Scaling any process can be very difficult.
- SAFe is a big proponent of having all hands meetings for portfolio discussions.
- With SAFe, there is a need to train everybody. This can become complex.
- SAFe is still a relatively new codified approach.
- Self-organization vs. Command and Control: SAFe supports self-organization as a major component of its values but its processes are beyond the scope of team level engagement.

(The Horror Of The Scaled Agile Framework, 2012)

Conclusion

The Big Picture was introduced as basic requirements for organizations to leverage in the management of software requirements for an Agile and lean environment. This framework uses a minimum number of roles, practices, and artifacts that are necessary for a team to be effective and then expands these to a program and portfolio level with each scenario providing the leanest approach to manage software requirements and allow teams of teams to build larger systems.

Chapter 14

The Waterfall Model

The Waterfall Model represents one of the pioneer approaches to software project management life cycles. It is also known as the linear sequential model. This was a project management life cycle that was introduced to the software development community to ensure that projects adhere to the triple constraints. Additionally, it is a project management life cycle that is widely considered easy to use and understand.

In past history, the majority of organizations and delivery teams relied on the Waterfall Model as the preferred project management life cycle. However, the introduction of Agile methodologies changed this position. Waterfall is still effective and used, but today people either use variations of the Waterfall Model or combine it with other models to create a modification that can fulfill the needs of their environment.

Many improvements and enhancements have been incorporated into this project management life cycle through its repeated use by several different people and companies. With Waterfall the next phase does not begin until the previous one is completed. The main objective of the Waterfall Model is to get early feedback to incorporate changes as soon as possible. However, the success rate of this model is relatively small compared to other project management life cycle options and, like everything else, it has risks associated with it.

What is the Waterfall Model?

The Waterfall Model is the earliest model in the history of software engineering that is applied in project management. Many subsequent project management life cycles used the Waterfall Model as a starting point. Waterfall was specially designed for software companies and they widely used it to achieve efficient and effective outcomes.

With Waterfall, phases do not overlap. A subsequent phase can only start if the previous phase has completed. Thus, it forms a sequential process to perform project activities. This is illustrated in the figure below. Historically, the life cycle performs quite well when quality is the main focus of the project.

This is an intensive project management life cycle that requires a lot of documentation in each phase of delivery. The documentation assists in the decision making process as to

whether or not the next phase should begin. In addition to the requirements document, there are many other project related documents that must be created. The list below details many of the documents required for a project (DeGrace & Stahl, 1990).

- Business Requirements Document.
- Problem Statement.
- Project Definition.
- Requirements Document.
- Preliminary Project Plan.
- Preliminary Design Document.
- Programmer's Manual.
- User's Manual.
- Test Plan.
- Installation Document.

Well documented requirements and designs help to identify issues from the very start of the project. However, this project management life cycle is rigid and does not handle change management well. Compared to other life cycles, Waterfall often requires more rework and costs more. As such software quality can be adversely affected by issues arising in the later stages of testing.

At the very start of a project, with this life cycle, complete and thorough software and system requirements are created that provide needed information for flexibility in design. After design, execution is started beginning with the coding phase. The final stages at the end of the project are testing and maintenance (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

History of Waterfall

As mentioned, the Waterfall Model was one of the most popular project management life cycles in past years and gradually was modified several times to enhance its performance.

The life cycle originally came into existence when projects started becoming more complex and could not be easily delivered by simpler execution processes. It is often difficult for a project team to look at the end result of a project in its entirety. This is why project management life cycles are used. They strive to break up the project into manageable components. Waterfall is no different in this regard.

In 1970, the concept of the Waterfall Model was debuted in an article (Royce, 1970). However, the "Waterfall" name was not used at that time. The paper was just meant to suggest a simpler idea for the delivery of projects. Royce simply defined phases that included requirements analysis, design, code, testing, and deployment among others. However, despite its simplicity, it does involve some amount of risk. Royce stated that the model "is risky and invites failures" and he also gave five suggestions to improve the life cycle to an incremental release model so that risks could be reduced (Kessel, 2013). The five suggestions he made are listed below.

Program Design Comes First

It kind of goes without saying that design is a critical stage. This is the first chance to identify risks and prevent project failure.

Document the Design

Documentation of the design with the use of Waterfall is highly recommended so that complete and clear requirements can be communicated.

Do it Twice

This is where incremental releases are introduced. Instead of attempting to deliver the whole project at one time, Royce advocated to build it in small incremental releases (Royce, 1970).

Plan, Control, and Monitor Testing

Testing of code, at least once, early in the life cycle is highly recommended. Additionally, it is also recommended that testing is completed by somebody other than the developer.

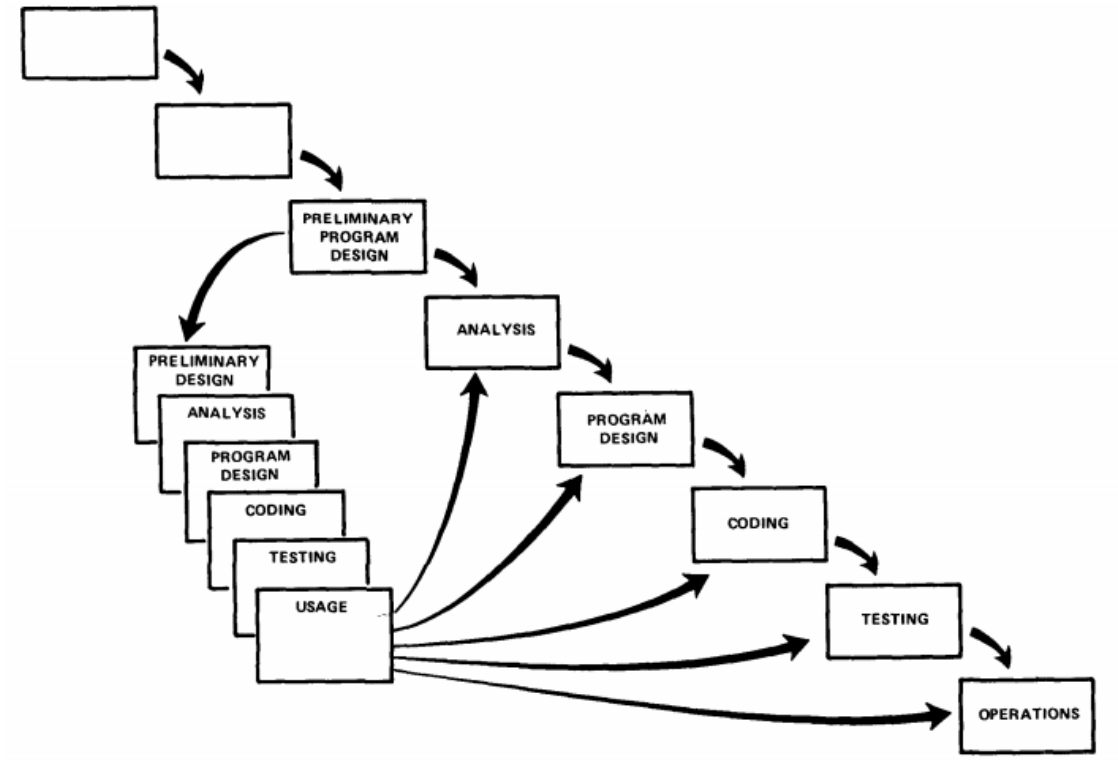


Figure 14.1 Illustration of Do it Twice Approach

Involve the Customer

The participation of the customer in the project was also emphasized, which is widely adopted today. This participation must be throughout the requirement and design phases at a minimum.

All said though, people adopted the initial Waterfall Model for a long time instead of the modified incremental release life cycle. In spite of its lower project success rate, this model is still widely used today.

Steps of the Waterfall Model

The figure above shows the names of the phases of the Waterfall Model. Each phase has its own set of activities. Below we will describe in detail the activities of each of those individual phases of the Waterfall Model (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

1. **System Requirements:** The first phase of Waterfall is system requirements gathering and analysis. As the name would imply, in this phase, all the requirements related to the hardware, tools, and other essential components of the

final deliverable are gathered and analyzed. A successful kickoff of a Waterfall project is essential with this project management life cycle.

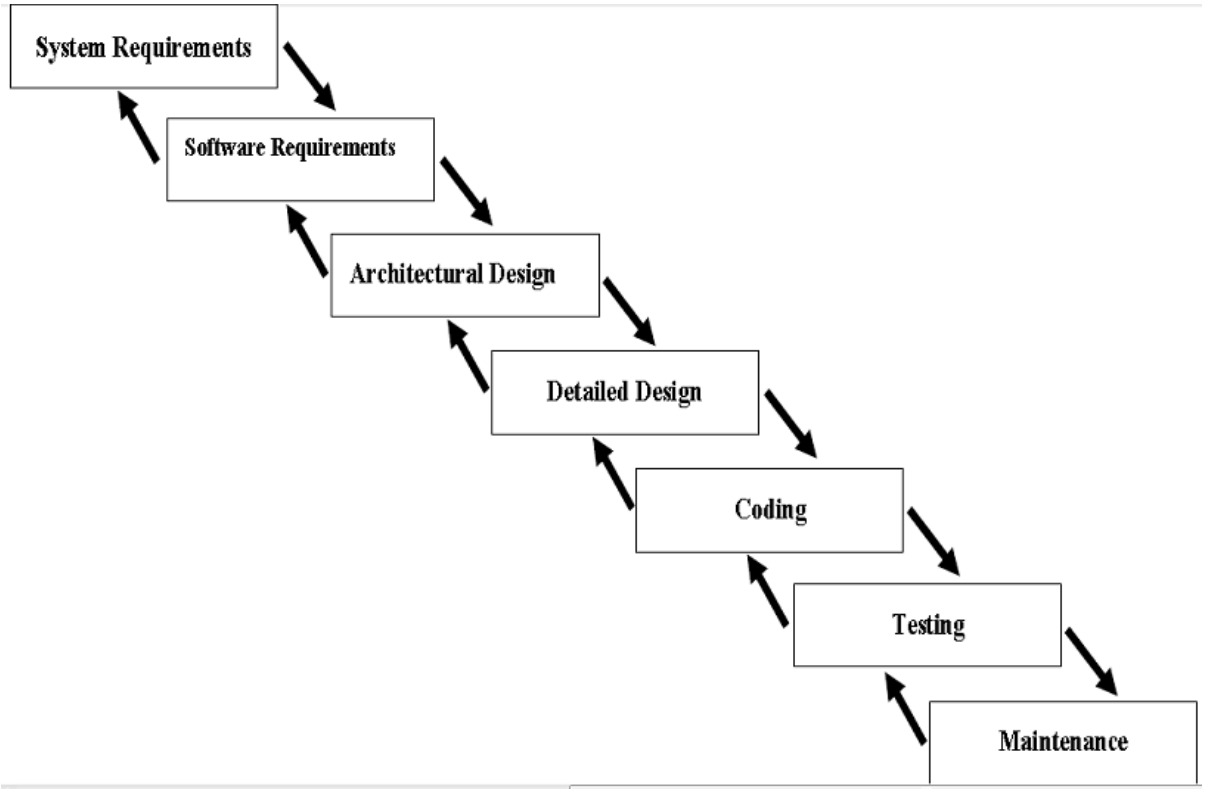


Figure 14.2 Illustration of Waterfall Project Management Life Cycle

The hardware consists of the requirements related to the type of computer platform and operating system where the final product will be installed. This includes any peripheral devices like a mouse, keyboard, speakers, printer, and other hardware components. Any software tools are also planned for in this phase, which includes the requirements related to databases, external applications, and other software components. All of these requirements are completely defined in the final requirements document.

2. **Software Requirements:** In the software requirements gathering and analysis phase, all the functional and non-functional requirements of the final product are collected and analyzed. This also includes any constraints for the final product. These requirements include all of the decisions made related to the user interface, design, performance, behavior, and anything else deemed pertinent. As in the first phase of the process, all of the software requirements are clearly defined and documented.

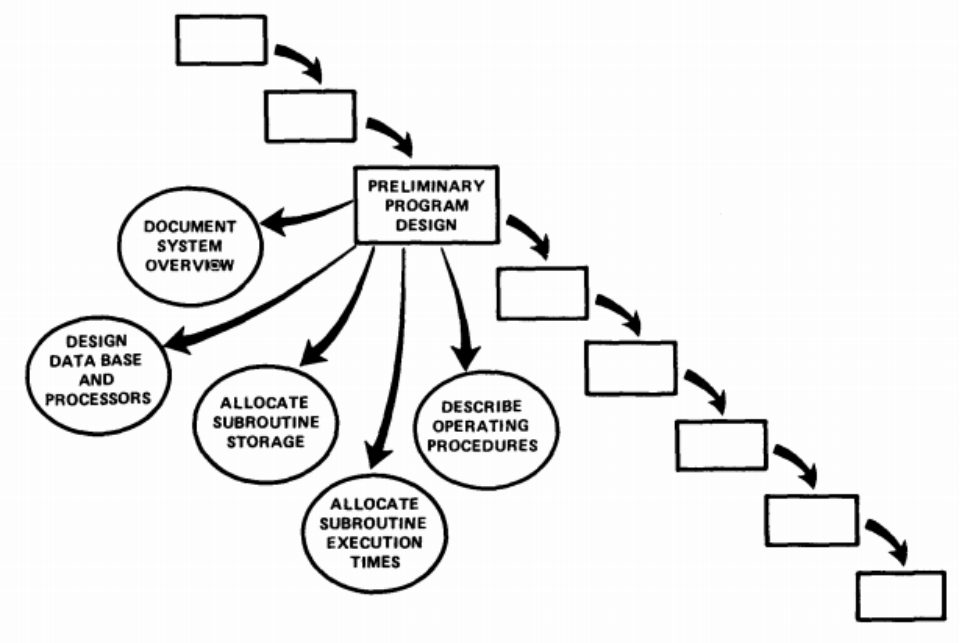


Figure 14.3 Details of Preliminary Design Process

3. **Architectural Design:** In the architectural design phase of Waterfall all the base requirements begin to be laid out for execution. The architecture is the foundation for a product much like the foundation of a house. In this phase, different attributes of the final product are carefully evaluated in respect to how they work with each other. However, the structure of the individual components themselves is not detailed in this phase of the project.

4. **Detailed Design:** During the detailed design phase the delivery team takes a closer look at the individual components of the final product. This is a granular approach that first defines the interaction of all components in the architectural phase and then looks at the individual components in detail during this phase. All of this is in preparation of the subsequent phase, which is coding.

5. **Coding:** The coding phase is where the rubber meets the road. In this phase the detailed design serves as the input to implement the individual components of the final product. The final product is reduced to work packages for implementation and then unit testing is completed on each individual unit of the product that will later be integrated to form the end product. The unit testing is done to verify that all the functional and non-functional requirements have been fulfilled.

6. **Testing:** In the testing phase the activities are either completed by the team members or by another party. However, it is highly recommended from the perspective of quality that identification of errors is completed by a person or team separate from the delivery team. This ensures separation of duties and that the delivery team does not try to fix their own errors on the fly thus circumventing the project management life cycle process. This is imperative because after testing the software is released to the market or end user.

7. **Maintenance:** The maintenance phase is the last phase of the Waterfall Model. In this phase, the problems that ultimately arise after product release are identified and documented with proper customer feedback. Problems are addressed in the maintenance phase and ultimately resolved in later releases of the product. The sole purpose of this phase is to provide a flawless product to the end user. In many organizations, it is said that this phase is never ending because given the rapid changes in technology the product is never fully completed. In the end, an organization must manage their product roadmap.

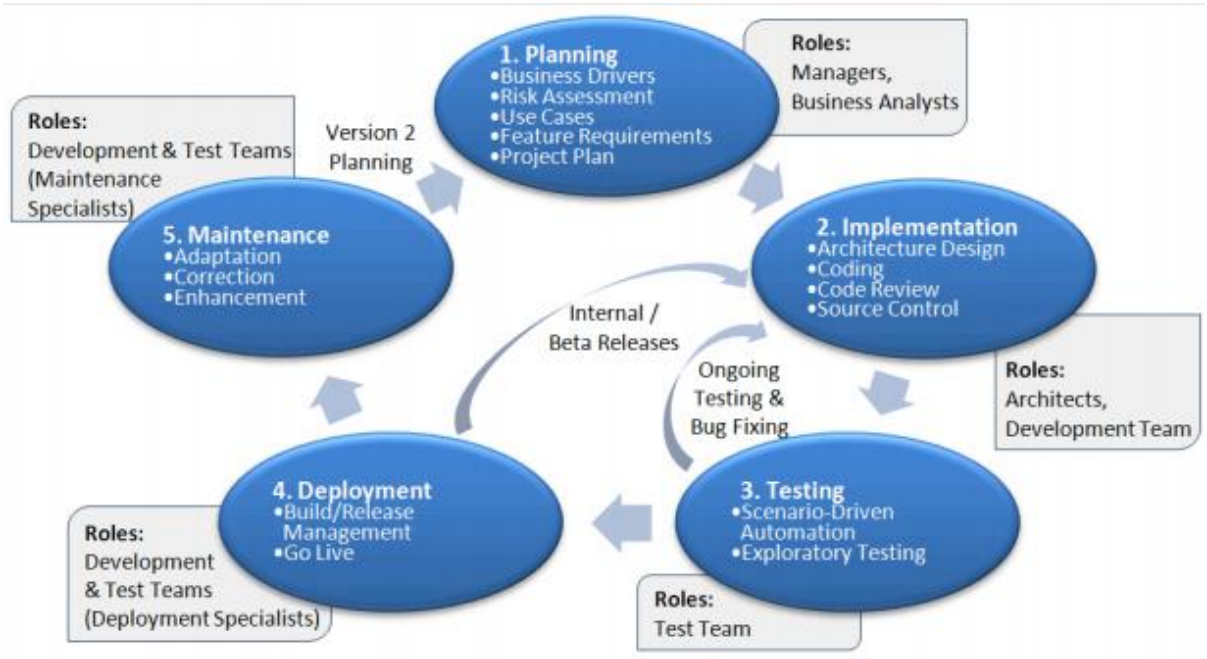


Figure 14.4 Abridged Illustration of Waterfall with Roles

Factors for Choosing Waterfall

There are generally several factors in considering a life cycle for a project because every project is unique and has its own set of requirements. Some of those factors are listed below.

- Are the requirements dynamic, well defined, or vague?
- Is the project scope consistent?
- Does the project use new technology?
- Is the project large or small?
- Are required resources and expertise available to support the project?

Strengths of the Waterfall Model

Many software companies widely use this development process and provide some of the arguments listed below in favor of the Waterfall Model.

- The model places emphasis on documentation of requirements and source code, which helps the project team understand the effort.
- The model runs smoothly in a linear sequential manner and provides a structured approach to project management.
- Each phase is easy to understand and milestones are easily identified.
- Waterfall performs best when requirements are stable.

Weaknesses of the Waterfall Model

There are people on the other side of the fence when it comes to Waterfall though. Some of the arguments against the use of Waterfall are listed below.

- Midstream changes in requirements can result in lost time and effort.
- Estimating time and effort for each phase is very difficult.
- The life cycle has limited management and planning control.

- Risk management is not prevalently included in the project management life cycle.

Pure Waterfall Model

Some statements in the project management community have referred to a "Pure Waterfall" model. In the Pure Waterfall Model, as with others we have already discussed, no phases overlap. As with other versions, this model performs well when the requirements are not ambiguous and are well documented. It is of course helpful if all the technical requirements related to architecture and tools are also well understood. The following diagram shows how the Pure Waterfall Model advances from one phase to another (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

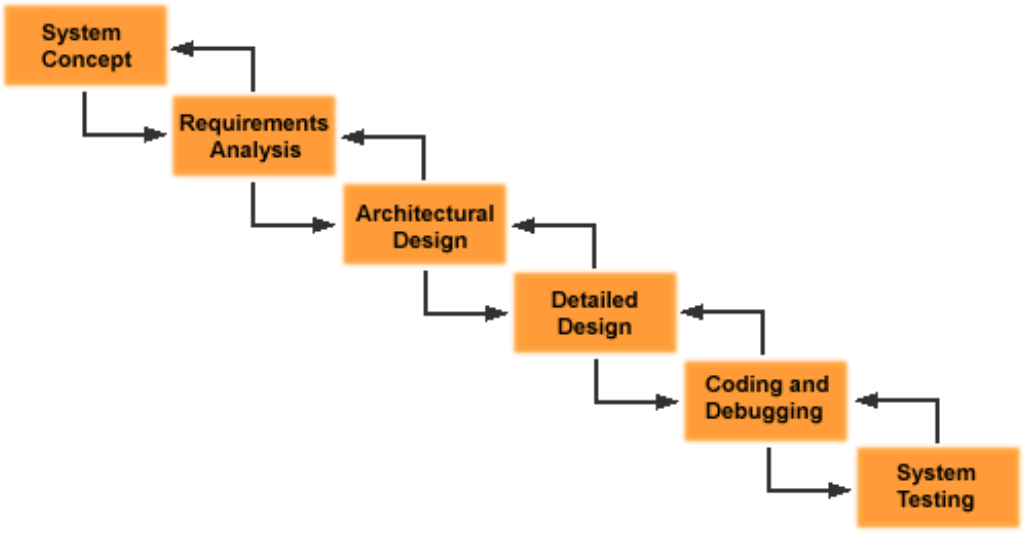


Figure 14.5 Pure Waterfall Model

Pure Waterfall Model Steps

The Pure Waterfall Model consists of the steps below (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

1. System Concepts.
2. Requirements Analysis.
3. Architectural Design.
4. Detailed Design.

5. Coding and Debugging.
6. System Testing.

If requirements are clear and stable then it is easy to identify issues early in the project. In the Pure Waterfall Model, all the planning is done upfront. Additionally, throughout the whole project management life cycle, documentation is created that assists the progress of the project.

Strengths of Pure Waterfall

- This model is good when the project team is inexperienced because it is well structured.
- Strict change control helps identify potential problems early in the project.
- This project management life cycle can be used to complete complex projects in an orderly fashion.

Weaknesses of Pure Waterfall

- This life cycle is not flexible and it cannot accommodate changes easily.
- This model does not perform well in rapid development environments.
- All phases, except the last phase, require documentation.
- With this project management life cycle there can be difficulty in documenting requirements in detail before design.

Modified Waterfall Model

The Modified Waterfall Model generally progresses the same way as the pure Waterfall Model because the basic activities are the same. That said, with the Modified Waterfall Model, there is less emphasis on documentation. However, the rest of the process should be executed as if it was Pure Waterfall. This model can be additionally modified by using other approaches like prototyping, splitting into sub-projects, risk reduction spirals, joint application development, and more (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

Most of the problems that arise with the Pure Waterfall Model are due to the stringent sequential nature of the phase progression. However, in the Modified Waterfall Model the phase progression is allowed to be altered. This means the phases can overlap each other, but only when it is for the benefit of the project. Additionally as previously suggested, this model can be divided into sub-projects after the architectural or detail design phases when needed. Furthermore, risks can be reduced by using the Spiral model in conjunction with the Waterfall process before starting any phase of the project.

Different Waterfall Models

- **Waterfall with Overlapping Phases / Sashimi:** The Sashimi model was created by Peter DeGrace (DeGrace & Stahl, 1990). This Modified Waterfall Model allows for continuous phases so that the phases can overlap as needed. The model also reduces the need for well-defined documentation (McConnell, Rapid Development: Taming Wild Software Schedules, 2010). If detailed documentation is not required, then this can be an effective version of the Waterfall project management life cycle.

- **Waterfall with Sub-Projects:** In a rapid development environment a project can be split into sub-projects. These sub-projects are executed independently and later integrated to form the complete product. It is often easier to implement the sub-projects independently. Generally, the best way to divide the project is to split it up after the detailed design phase (McConnell, Rapid Development: Taming Wild Software Schedules, 2010).

- **Waterfall with Risk Reduction:** In this Modified Waterfall Model, the Spiral model is added for risk reduction. This technique was used by DeGrace and Stahl, who also named this model “Whirlpool.” The Spiral approach is added before starting the Waterfall process and includes all activities to manage risk. When using this Modified Waterfall Model, it is not necessary to document all the requirements before architectural design (DeGrace & Stahl, 1990).

Strengths of the Modified Waterfall Model

- It is more flexible than the Pure Waterfall Model.

- Detailed documentation can be reduced.

- Using Waterfall with sub-projects allows the implementation of the smaller and easier components first.

Weaknesses of the Modified Waterfall Model

- Tracking is more difficult due to overlapping phases.
- Interdependencies between sub-projects can create problems.
- There can be a high potential for risk due to team member assumptions (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

The Waterfall Model is used widely by software development companies, however, as with all project management life cycle models it also has its problems. One key weakness with this life cycle is its inflexibility to change. Serious problems can occur later in the testing phase that could be difficult to resolve and even cause project failure. However, Waterfall does work well for companies with small projects.

Advantages of the Waterfall Model

The Waterfall Model has the advantages that are detailed below (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

- It is an easy process to understand and implement.
- Specification documents are well defined and other detailed documents are created throughout the whole project management life cycle.
- Each phase has fixed deliverables and clearly defined milestones.
- The phases of the Waterfall Model are not overlapped and are completed one by one.
- Each phase of the project life cycle model has its specific deadlines and milestones that are required to achieve project delivery smoothly.
- When quality is the main focus of project delivery, Waterfall works very well.

- Waterfall performs well for small scale projects when there is no ambiguity in requirements.

Disadvantages of the Waterfall Model

The Waterfall Model is also known to have the disadvantages listed below (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering, 2010).

- Waterfall can have increased project risk and can produce uncertain results.
- Waterfall is not necessarily appropriate for complex projects and this is especially true for object oriented projects.
- Change management during the testing phase can be difficult.
- Risk management can be very poor with Waterfall.

V- Methodology

In some ways, V-Methodology is an extension of Waterfall. This project management life cycle gets its name from the diagram of the process. The process is represented as a V as seen below. You start at the top of one leg and work your way down and when you get to the bottom you work your way up the other leg. Additionally, the "V" in the name stands for Verification and Validation (What is V-model - advantages, disadvantages and when to use it?, 2012).

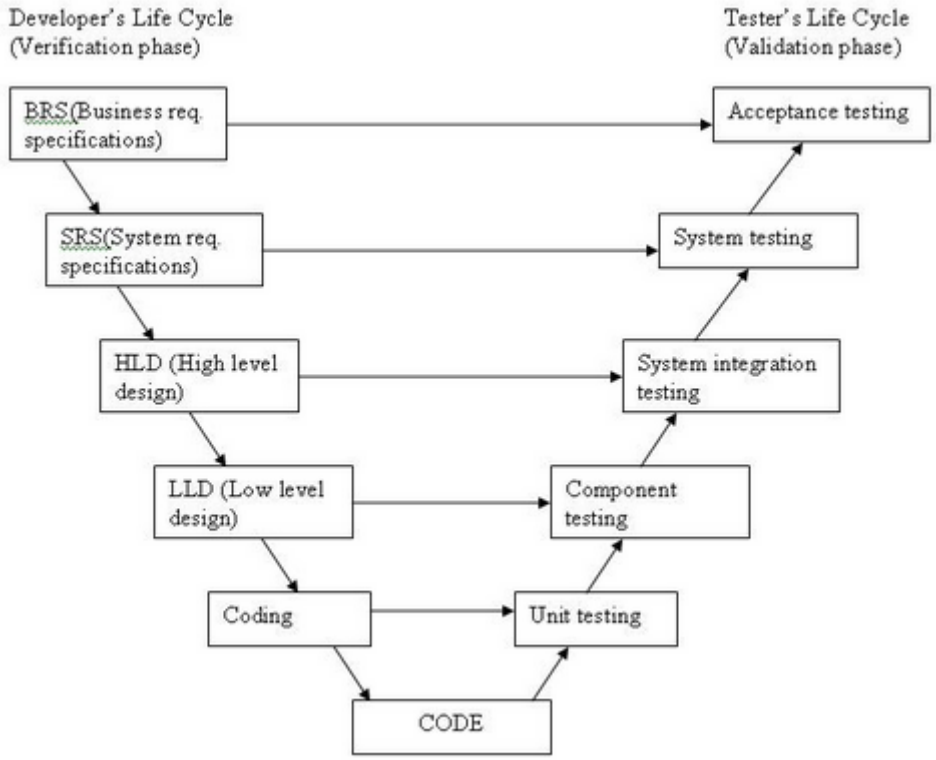


Figure 14.6 V-Methodology

Developers working on large scale product development projects normally opt for incremental methodologies like Agile. However, this methodology has been shown to be preferred when the requirements are clear to everyone and the architecture is decided beforehand (Srinath, 2009). With this process, testing and review procedures are also performed at the end of the project and only once.

Execution is completed in V like fashion. Development takes place on the left side similar to the Waterfall Model and testing takes place on the rising right hand side (Changede, 2013). Verification and validation work on a one-for-one approach giving them both equal weight.

History of V-Methodology

The V-Methodology was first defined by Paul Rook in the 1980's. It was basically meant for improving the processes in software development (Reddy, 2013). The first instance of V-Methodology was seen as a project management approach by the German government and named "Das V-Modell" (V-Model, 2016). Mostly it was named so because of the V shape the steps in the model take. The rest of the world recognizes the description of this model as detailed in the International Software Testing Qualifications Board (ISTQB) Foundation's Syllabus that is meant largely for software testers.

Phases in V-Methodology

Regardless of the differences in graphical models of the V-Methodology, the basic phases are the same. Any project management life cycle that works on the principles of the V-Methodology proceeds through the phases detailed below.

Requirement Analysis and Test Planning

Requirements and expectations for the final product are gathered from all the stakeholders. This includes interviewing end users to find out what they want. This data becomes the foundation of the project and even vague details are discussed in this phase. The system designers kick off their work based on the requirement analysis document. In addition to these activities, the software requirement specification (aka SRS) and the testing procedures are documented in this phase.

High Level Design

A high level design, of course, must be completed before moving on to low-level design details. As such, a blueprint for the final product is documented in accordance with all specification documents.

System Design

The next step in the development process is to design the final product. The project scope must be completely understood if all user requirements gathered in the previous phase are to be delivered. Additionally, a general overview of the software product is completed in this phase. Furthermore, relationship diagrams are also prepared to demonstrate how different entities will interact with each other and test procedures are completed.

Architectural Design

High level design operations like architectural design and software architecture are completed during this phase. A specific architecture is decided upon and technical details of the design relationships and underlying schema are resolved in this phase.

Module Design

The overall project work is ultimately divided into manageable modules so that the developers and programmers can complete the effort systematically.

Construct Phase

The main work on the project starts here. At this point all the design modules are defined and actual work needs to be performed. This is where code and test reports are completed and compiled.

Unit Testing Phase

After the development work is complete, unit testing is performed on each piece of code. Essentially, validation is completed to make sure that the product conforms to the requirements that were captured.

Final Release

After testing is completed the final product is released for customer acceptance. Additionally, software quality assurance (SQA) requirements and audits are carried out to make sure that the final release conforms to all standards. Furthermore, technical reviews are performed to assure conformance with all standards (Srinath, 2009).

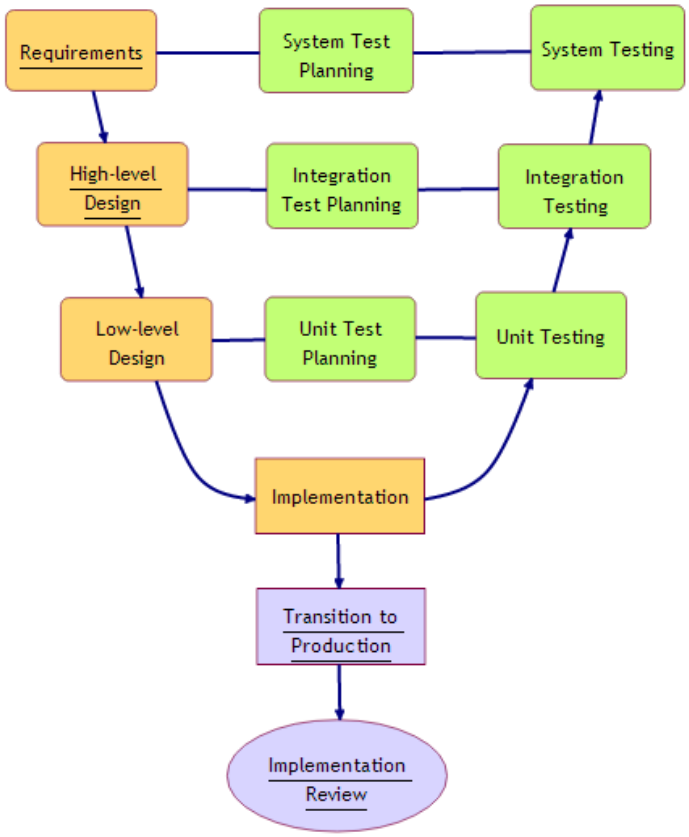


Figure 14.7 Example of Software Development Using V-Methodology

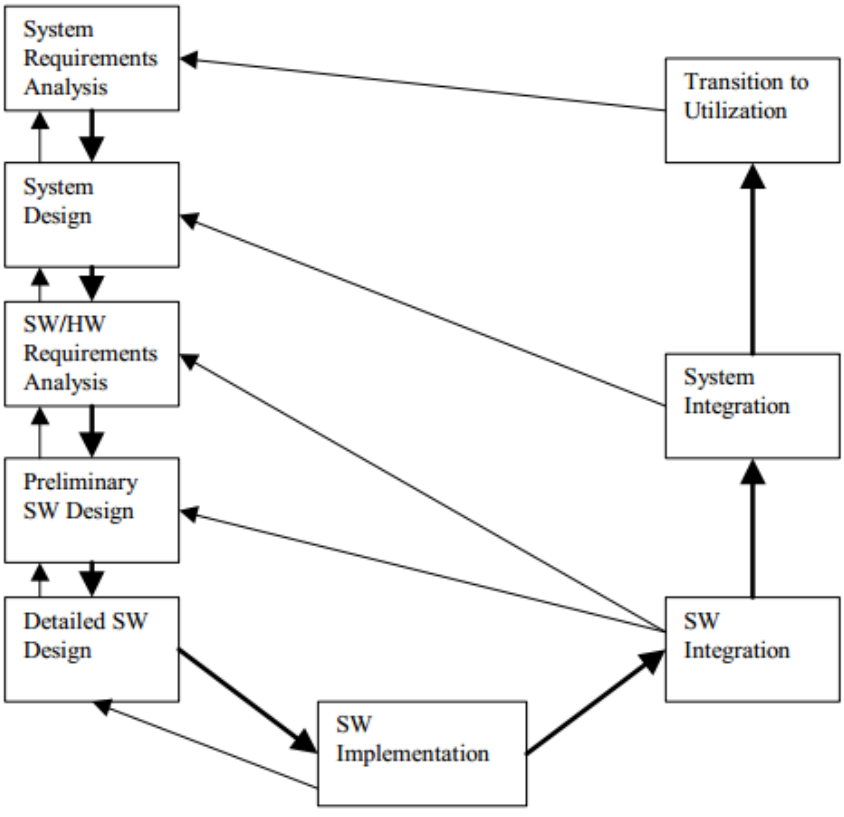


Figure 14.8 Additional V–Methodology Model

V-Methodology Today

Most organizations have abandoned the use of the V-Methodology and adopted other life cycles like Agile. Yet, there are still some organizations that have enhanced the V-Methodology life cycle to include support and maintenance phases. This allows the project life cycle to be lengthened and reduces the need for a sequential series of projects.

Advantages of V-Methodology

Identified advantages of the V-Methodology project management life cycle can be found below.

- Project managers normally prefer V-Methodology for its simplicity. It is a completely no fuss method and is easy to follow in a step-by-step manner.
- Numerous testing activities are performed before coding is initiated.
- Once requirements are documented execution is much faster than with many other models (What is V-model - advantages, disadvantages and when to use it?, 2012).
- Since requirements are completely documented before any work begins, it is generally easier to select the right tools for execution.
- The V-Methodology project management life cycle is well suited for testing.
- V-Methodology performs well with estimates for fixed price projects.
- V-Methodology supports multi-layered designs.

Disadvantages of V-Methodology

Identified disadvantages of the V-Methodology project management life cycle can be found below.

- The V-Methodology is generally considered better for project management than software development so many developers resist its use.
- The V-Methodology can become complicated with projects of greater complexity or scale.

- The V-Methodology is not an incremental release project management life cycle and as such this can increase project acceptance risk.
- The scope of this life cycle is limited to the project level and cannot be expanded to the organizational level like some other life cycles.
- Post release bug fix and maintenance are not part of this project management life cycle. Instead any additional effort requires the kickoff of another project.
- The rigid nature of this project management life cycle inherently resists changes.
- Changes can significantly set back a project by creating a return to the requirements phase.
- Since V-Methodology is not an incremental release project management life cycle, phases can't be repeated as part of normal process.
- The V-Methodology project life cycle has been criticized as having a greater emphasis on theory and less on execution.
- Given the evolution process of this life cycle over the years, there is often not a universal vocabulary amongst the project team.

Chapter 15

Prototype Model

Prototype has a relatively simple definition. It is merely a sample or first impression of what a product may look like before it is finally released. They help to understand if the conceptual idea behind a product is realistic and achievable. Prototypes are important for providing a real time experience with a product while it is still in development. The closer a prototype is to the desired final product the less likely there will be design flaws with the end result. Ultimately, the purpose of prototyping is to evaluate an idea and its functional operations to understand and address problems and issues that may occur in the final product.

The Prototype Model has become an important part of the design process for several types of goods including electronic and software products. System analysts and end users are generally involved in testing prototypes to enhance the product's functions and ensure successful delivery. Prototypes are the initial face of what the actual product will become and they often offer a more practical approach to theoretical modeling. Nowadays, while the Prototype Model is considered a project management life cycle of its own, many other project management life cycles incorporate a prototyping phase between the formalization and evaluation phases to analyze the validity of an idea.

With the Prototype Model, a fully functional and complete version of the product is ideal for testing and evaluation in real world scenarios, however, there are times when prototypes are completed merely to evaluate the form, feel, and function qualities of a product or various other aspects of the desired end result. Results from the testing effort are then used to re-calibrate the delivery team's actions for completing the final product. By now, it should be obvious that prototyping is an effective technique for understanding defects and issues in a product before its final release.

In software engineering, the Prototype Model, or at least a prototyping phase, can be an extremely useful part of the delivery process. As with any other effort, developers diligently work on prototypes to provide a real experience during testing because this is work that can be used for the final delivery and is not wasted. A prototype is basically like a dummy product and the benefits of prototyping are certainly endless (Sabale & Dani, 2012).

Developers and engineers rely on prototyping when requirements from clients or end users are incomplete, unknown, or ambiguous. Through the use of prototypes, project requirements and features become clearer and better understood. Thus, it can be said that prototyping is more like a trial and error approach to product development and the prototype assists in the discovery process for unidentified or unclear requirements.

When customers or end users are engaged with the prototyping effort, they will often approve the prototype as is or ask for changes, improvements, or modifications. This information will then guide the rest of the delivery team's efforts. Prototypes, as with the final product, are evaluated against a requirements checklist to validate and confirm that all features, functions, and attributes have been successfully delivered. For large and complex projects where iterative cycles are required and customer needs keep evolving, the Prototype Model is perfect and one of the best approaches to cope with complications.

History of Prototype Model

Not a lot can be said about the history of prototyping. Beginning with the first steps a person takes in their life, experiences become a trial and error process. Even Thomas Edison made thousands of light bulbs before he was finally successful. All the ones prior to the final version of the product could be seen as prototypes. What is known about the history of the Prototype Model, though, can be found below.

The word prototype can trace its origin to the Greek word "prototypon" that means a primitive or the crudest form of anything. Another possible Greek origin of the word prototype has also been suggested. Derived from the word "protos" meaning first and the word "typos" meaning impression (Wikipedia). All these words, of course, tend to point towards the first or the most original form of any object. Improvements in the object come later.

For example, in the manufacturing industry the prototyping concept was first introduced by Herbert Voelcker who was an engineering professor. He studied the possibilities of using prototypes that could be further enhanced using automated machine tools that emerged in the sixties (Rounds, 2008). Nowadays, every industry that is involved with engineering, manufacturing, and building any product uses this approach. Prototyping may take a longer time, but it reduces the possibilities of risk and failure.

Phases of the Prototype Life Cycle Model

The Prototype Model has various phases. Below each phase of this project management life cycle is discussed in detail.

- **Basic Requirement Identification:** The initial phase of the Prototype Model is where requirements definition activities take place. End users and management discuss what the product needs to look like and its basic functionality. At this stage, security and performance requirements are generally ignored usually because the main focus is more on the overall concept and user interaction with the product.

- **Developing the Initial Prototype:** The next phase is where an initial, but not fully functional, prototype is built to get an idea of whether or not the project team is on the right track. This is where end user interaction is showcased. At this stage it is common for the prototype to only provide a look and feel that is similar to the anticipated end result.

- **Review of the Prototype:** Here management and customer feedback about the prototype is received and their reviews are recorded and evaluated. All other stakeholders of the project give their reviews as well. All reviews, feedback, and criticism is organized and recorded for further enhancements and improvements before completion of the final product.

- **Revise and Enhance the Prototype:** At this phase, an in-depth analysis is conducted. The reviews and feedback collected during previous phases are now investigated completely to analyze the feasibility of every possible dimension that could or should be incorporated into the finished product. Feasibility and various factors like time, budget, resources, and deadlines are analyzed against the project requirements. All accepted and mutually agreed upon improvements are then implemented in the product.

- **It Goes On:** The prototyping process and testing of the results continues until all stakeholders are satisfied with the product's final look and operational capabilities. At this point, product deployment takes place. Once the product is deployed to the field, then the maintenance phase of the project management life

cycle starts and lasts until any other improvements in features and functions are required or desired (Saini & Kaur, 2014).

Effects of the Prototype Model

The use of the Prototype Model for software development resembles many software centric project management life cycles covered in this book. All life cycles have their advantages and disadvantages and where many software life cycles are weak, prototyping is often found to be strong. The component that causes the Prototype Model to resemble many software models is its iterative nature, but this alone should not be the only aspect to compare with other project management life cycles.

When it comes to the software environment, prototyping is used when it is almost impossible to gather all of a project's requirements at one time. This can be because sometimes the project team fails to record and understand all the client's requirements in just a few initial meetings or just that the client simply doesn't completely understand what they want or need. In situations like this, prototyping assists in the development process and the identification of requirements. This approach is not specific to the software industry, however, and can be used in many environments, but it should be understood that a project management approach based on lack of planning and multiple prototypes becomes more and more expensive with each and every prototype.

Prototyping doesn't just help when requirements are vague and transitional though. Sometimes it may be clear what is desired, but the question is how to deliver the requirements or whether or not the requirements can be delivered at all within the constraints of the project and the technological solutions available. The space age of old, and even today, is a very good example of this. Nobody had ever landed on the moon before or even been in space, so there were many unknowns in need of prototyping. The figure below shows one version of the Prototype Model.

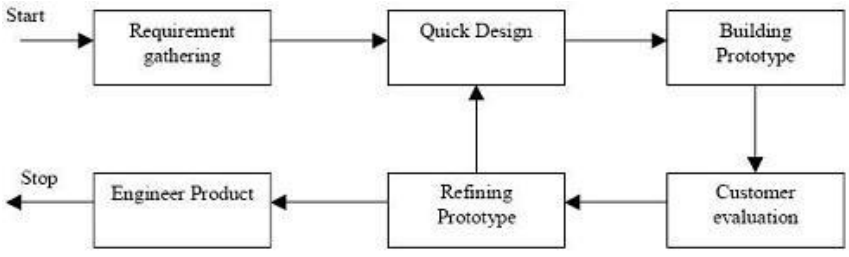


Figure 15.1 Prototype Model

However, even with just one prototype for a project, arguments can be made that prototyping is an expensive process. It really comes down to risk. The project team must decide whether it is more risky and expensive to attempt the project without a prototype or less risky. Returning to the space example, it was a very risky project that could have quite easily cost lives.

Combating cost arguments these days has become easier with different prototype technologies. These include 3D printers and reduced functionality prototypes that can be completed with less expensive materials. It should be understood that a prototype is never a complete product. It always lacks some detail and features as it is only an early stage model or a sample of the actual product. However, as stated, the process is still extremely successful in achieving a final product with fulfilled requirements and completed functionality.

In addition to all the previously stated benefits of prototyping, during the process, designers typically gain more confidence and experience in how to produce and deliver the final product. This confidence serves to reduce risk adverse behaviors and encourages designers to take the next steps toward completion of the project. This is yet another aspect of the project learning curve that assists with reduction of costs. Therefore, as with all project management life cycles, the Prototype Model has its shortcomings, but its benefits far outweigh its disadvantages.

In the below diagram of the Prototype Model, it is clearer how prototyping can be compared to software project management life cycles like the Waterfall Model. However, with this example, which assumes client involvement, the client and designers interact at each and every stage. The popular belief among those who are advocates of prototyping, is that the more the customer is involved, the greater the chances are of rejection of the final product.

Prototype Model Approach

Many researchers and engineers alike believe that prototyping is an exceptional approach because it resolves many issues effectively. The use of the Prototype Model in the software industry ultimately came into the limelight when, yet again, the software developed according to the Waterfall Model showed shortcomings and major issues. The delivery team then discover that the reason for failure came from incomplete and unknown user requirements that were overlooked with the use of the Waterfall Model (Munassar & Govardhan, A Comparison Between Five Models Of Software Engineering , 2010).

Moreover, user requirements are kept static with the Waterfall Model. However, with many other software development approaches, the requirement process is included in initial steps, but is not final until the end of the project.

In this environment, the gap between the developer's understanding and the actual requirements is generally thought to be the major reason for failure of any product development process. Using prototyping as an iterative approach to design eliminates this issue. This is because including any incremental release loop within a project management life cycle has the effect of eliminating the need to go back and forth between two major consecutive stages of a product delivery process. This is not to say that after the release loop that requirements can't be frozen to move on with the delivery process, because in fact they can.

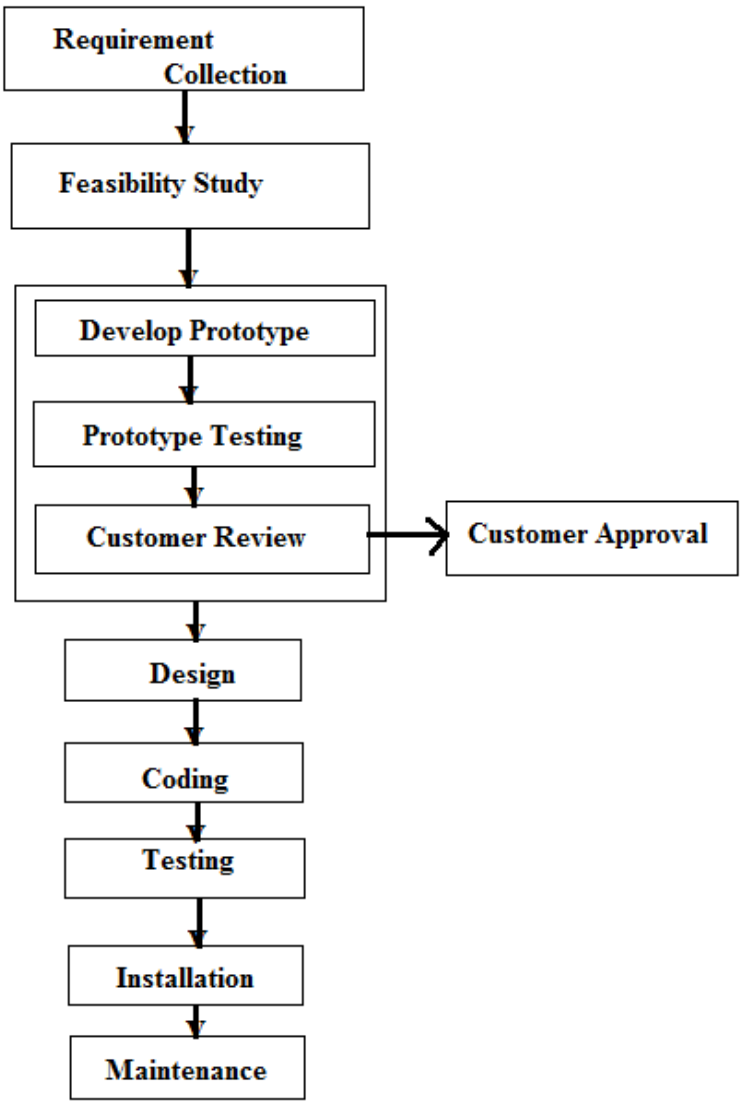


Figure 15.2 Another View of the Prototype Model (Prototype model of SDLC, n.d.)

Example of Prototype

As stated, prototyping can be used in multiple industries and in multiple ways. Let's take a look at a simple example involving an e-commerce website where the approach of prototyping works quite well. Using the Prototype Model, a team could develop and design the initial pages of a website incorporating all features including products, shopping carts, payment processing, and shipping. This could then be shown to the customer for review and approval. Upon approval, the team would complete the final product. Any changes required by the customer could, of course, be implemented into the final product.

Categories of Prototypes

There are two basic categories of prototypes. They are named horizontal and vertical. Both versions have different purposes. Below, they are discussed in detail.

Horizontal Prototypes

Horizontal prototypes are meant to display a wide range of features without fully implementing the features. End users and management get an idea of how the product will look as well as user interaction. This type of prototype is more focused on the look and feel of a product and pays less attention to internal features and functions. It gives a broad and easily understandable view of the product without addressing operational features and functions. It is simply more focused on business requirements. Horizontal prototypes are the kind of prototypes that can be confidently presented in a customer focus group or a sales environment for introduction of the product.

Vertical Prototypes

Vertical prototypes are quite the contrary to horizontal prototypes. Their focus is to implement a small set of features nearly 100% for review by management and end users. This allows all stakeholders a deeper look at the features incorporated into the prototype. In short, vertical prototypes are merely a small subset of the desired final product.

When to use which Category

- The Prototype Model works well when the requirements of a project are unclear by management and end users, hence, focus groups leveraging horizontal prototypes works well in this environment.

- Since regular customer input and contributions increase the chances of project success, small sets of features frequently presented works well. This dictates the use of vertical prototypes.

Types of Prototypes

Basically, there are just a handful of different types of prototypes. They are detailed below.

- **Throwaway Prototypes:** These are all prototypes that are destined to be discarded and never become a part of the final product. Examples of throwaway prototypes are screen mock-ups and story boards. This variation of prototyping is also known as closed-end or rapid prototyping. With this type of prototyping every article provides a clear understanding of the requirements and the previous version is discarded (Scacchi, 2001).

- **Evolutionary Prototypes:** These prototypes are progressive in nature and they become a part of the final product when the development cycle ends. Such prototypes are called evolutionary prototypes. They continuously improve via a sequential process. These articles are also referred to as breadboard prototypes in some circles. They have a minimum of functionality at the beginning, which improves during the course of development.

- **Incremental Prototyping:** This process produces various multi-function prototypes and then integrates all the approved and functional prototypes into one complete system (maheshwari & Jain, 2012).

- **Extreme Prototyping:** This is a process sometimes used in web development. It has three phases. In the first phase, all html web pages are produced. During the second phase the use of service layer is constructed to simulate data processing. Finally, the fully functional user interface is developed to manage all the functional attributes of the system.

Prototype Model Tools

To properly implement the Prototype Model, an organization needs to determine exactly what are the best ways to meet the project requirements and, as such, which tools would best be of service. There are certain tools, standards, and training that are needed in

order to fully implement the Prototype Model and get maximum benefit from it. These tools include items such as 4th generation programming languages implemented to complete rapid prototyping and sophisticated CASE tools. Additionally, ColdFusion and Visual Basic are often leveraged because they are inexpensive and easy to use.

Case tools with supporting requirement analysis have proven to be helpful in an engineering environment, usually in large military organizations where prototyping is almost always necessary. There are various frameworks available in this environment including Bootstrap, Foundation, and Angular JS that help to create a proof of concept. These frameworks assist the project team in constructing prototypes faster.

Screen Generators, Design Tools, and Software Factories

In a software environment, screen generators are invaluable as prototyping tools and enable developers to quickly deliver a screen's look and feel for review by management and end users. This helps in visualizing the final product. Software factories are also readily available that offer ready to use modular components for project teams. Therefore, with all of these options, it should be understandable why many project teams prefer to prototype before jumping into delivering the final product.

LYMB

LYMB is an amazing object oriented development environment that is useful for applications combining graphics user interfaces, visualization, and quick prototyping. LYMB is useful with the Prototype Model because, like other tools, it provides cost and time savings as well as dynamic interfaces. LYMB is a development environment that has become quite popular in recent years because of its approach.

Non-Relational Databases

Through the use of non-relational data, end user prototyping has become valuable, simple, and efficient without any delays in the release of simulations. Additionally, business requirements are generally gathered more accurately and without ambiguities. Making this one more valuable tool.

Prototype System Description Language (PSDL)

PSDL is an interactive prototype system description language that made life for project teams less complex. The tool set used with this language is known as Computer Aided Prototyping System or CAPS. There are several prototyping software systems on the market that make the job easier, however, add in real time requirements with timing constraints that introduce implementation and hardware dependencies and PSDL shines.

Thankfully, PSDL efficiently addresses the above issues with the use of control abstractions that include declarative timing constraints. CAPS then uses this information to generate code and real time schedules as well as monitor timing constraints during prototyping. Furthermore, CAPS simulates execution in proportional real time relative to a set of parameterized hardware models. PSDL also offers default assumptions that facilitate the execution of incomplete prototype descriptions. Finally, PSDL assists with the rapid evolution of designs and requirements.

Simulation Software

Application or simulation software is amazing in its purpose and offerings. Developers are able to build animated simulations without writing a single line of code and it offers the opportunity for collaboration, testing, and real time user experiences. Project teams, management, and end users can all validate functional requirements without an in-depth engineering evaluation. With results oriented features like screenshots, annotations, and schematics, it should not be hard to see why many project teams prefer prototyping as a stage of the project management process.

While this approach may sound high risk and time consuming to some people, it has been proven to reduce costs and assist with greater understanding among stakeholders as to what is needed to produce the final product. Yet, there are still more specialized tools available for real time demonstrations and product evaluations to fully understand the full workflow of the product. Like all the tools in this section, this one serves to reduce time and save money.

Advantages of the Prototype Model

The benefits of the Prototype Model are numerous. Some of the major advantages of this project management life cycle are detailed below.

- When the end user is actively involved throughout the various stages of the delivery process, there is a greater chance of project success and customer satisfaction.

- Since a prototype is a functional version of the intended end result, the project team and end user gain insight on how the final version of the product should perform and are able to make adjustments before late stage development begins.

- Using the Prototype Model, defects and issues are identified earlier thereby reducing costs and saving time.

- The Prototype Model makes complex products easier to manage by allowing for sub-assembly testing throughout the project.

- The Prototype Method can serve to cast the project team in a good light with management and the end user by engaging them in the prototype process so that they can better understand the work being accomplished.

Disadvantages of the Prototype Model

Undoubtedly, the Prototype Model assists in paving the path toward product perfection, but nothing is ever flawless and this project management life cycle is no different. Below, the disadvantages of the Prototype Model are detailed.

- Prototyping can make a product seem simpler than it is leading to communication problems. This is because the prototype is a version of the final product that usually gives an impression to clients that only slight changes need to be made to complete the project, which is not necessarily the case. Sometimes things that appear to be minor changes on the surface are much more complicated.

- With client involvement comes a political balancing act and the same applies here. Many clients don't truly understand prototyping and when they finally see an artifact that does not include all the final product functionality they can begin to think that there is a lack of commitment from the delivery team to complete the project.

- Prototyping can become a crutch and lead to poor project management processes within the delivery team. This is because the team begins to feel that if the first

prototype doesn't work they can always make another after corrections. However, this just leads to project cost and schedule overruns.

- During the prototyping process, there is a chance of losing site of the goals for the end product especially if the prototypes are very good. This can lead to dissatisfaction when a final product doesn't deliver as planned.
- Prototypes cost money and are not necessarily cheap. This means often more times than not a project management approach with a prototype is going to cost more than an approach without a prototype and may increase project duration as well.

Chapter 16

Spiral Model

Developing software effectively continues to be an extremely difficult task at times. Even with the advent of new technologies and different methodologies for project management, software development has not been easy. Each software project is unique in nature and comes with equally unique problems. As this book shows, over time, several different methodologies have been constructed to deal with the changing nature of software design. The methodology we will focus on in this chapter is the Spiral Model.

The Spiral Model is yet another project life cycle that promotes the development of software in stages. It starts with requirements definition, moves on to design stages, implementation, evaluation, and then ultimately evolution. Prior to the Spiral Model, these stages were incorporated in the Waterfall Model by Winston Royce, which we discussed in detail earlier in this book. However, the Waterfall Model was felt to be unsatisfactory and clumsy because it didn't support changes effectively. The process promoted monotony and increments were planned in advance without any room for changes (Osterweil, 2011).

It is normally best suited for software that is well-defined and each requirement is clear in the minds of the development team. However, this is rarely the case as, often, software is unique with ever changing requirements. Also, the Waterfall Model is a document driven approach involving a great deal of documentation. This makes it difficult to use for the software project where requirements are not clear in the mind of the client and are constantly changing (The Spiral Model, n.d.).

To come up with a solution to this problem, Boehm made the Spiral Model. The Spiral Model incorporates the features of both the Prototype Model and the Waterfall Model making it more effective for large and complex IT projects. It is an incremental model that places more emphasis on risk analysis. The Spiral Model has four main phases and each of these will be discussed in detail later in this chapter (What is Spiral model- advantages, disadvantages and when to use it?, 2012). The phases are listed below.

- Planning.
- Risk Analysis.

- Engineering.
- Evaluation.

History of the Spiral Model

The Spiral Model is the work of Barry Boehm and was described by him in 1988 in an article called "A Spiral Model of Software Development and Enhancement." As stated throughout this book, the Spiral Model was not the first model to promote iterative development. The Waterfall Model and Prototype Model are examples of models that were already incorporating this approach. However, what makes it different from them is that it was the first model to explain why iterative development is important. It has been modified and enhanced greatly over the years and has proven to be very effective for large and complex projects (Spiral Model, n.d.).

As such, with the Spiral Model, Boehm proved that iterative development is the key to successful software development. He explained in his paper that iterations are driven by risk analysis and the risk is mitigated through the use of prototypes. He emphasized the need to develop software in increments as it opens up the opportunities for changes and better understanding between the development team and the client (Osterweil, 2011). Additionally, his work opened the door for newer and modified versions of the model such as the Win-Win Model (Novakouski).

What is the Spiral Model?

The Spiral Model is a Software Development Life Cycle (SDLC) technique combining the elements of both prototyping and design in stages and takes advantage of both top-down and bottom-up concepts. It is mostly used for large projects such as US military efforts. It is also used in business environments where the business goal is unclear and prone to change. Figure 16.1 below shows the spiral development process.

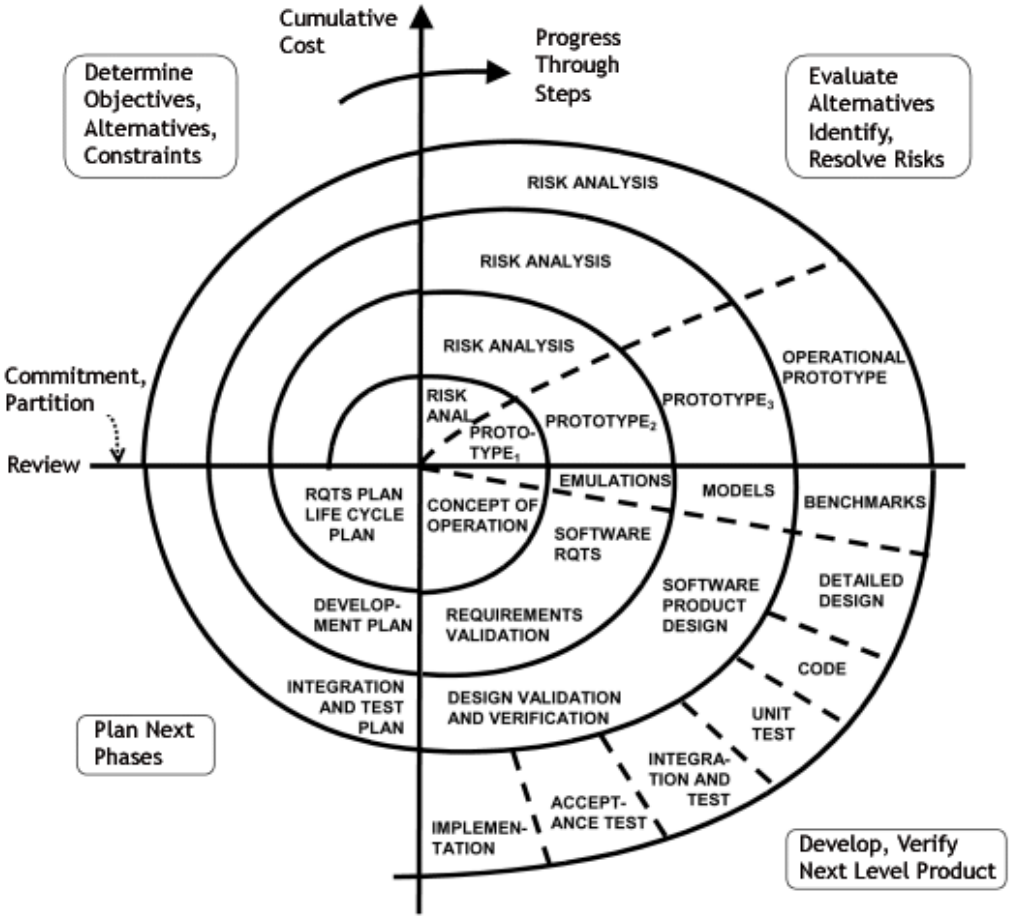


Figure 16.1 The Spiral Model

The diagram above details the four main components of the Spiral Model. It begins with the construction of a prototype. Each prototype developed serves a specific purpose and strives to create deeper understanding for the development team as well as the client. The final prototype is ultimately a blueprint of the code required for the completed software product. Thus, the delivery of the software is based on these prototypes. The requirements, design, implementation, and testing of the software is completed using prototypes as the main source of information.

As Figure 16.1 shows, the development process takes place in a series of incremental sequential cycles. Each cycle is connected to the previous cycle and includes the four main phases listed below (Osterweil, 2011).

- Determining objectives, alternatives, and constraints.
- Evaluating alternatives and risks.

- Developing next products.
- Planning for the next iteration.

The four phases of the model are used to carefully decide how to proceed further and how to address the complex tasks during the course of the project. The Spiral Model has a greater diameter and more scope for successful development of software. The Y-axis in Figure 16.1 represents cumulative cost that grows as the project moves on to later phases. However, the project does not only grow in terms of cost, but also in terms of risk, project knowledge, and artifacts produced. Risk mitigation strategies are determined at each phase, which creates an increasingly better understanding of the project risks (Osterweil, 2011).

In short, there are ultimately two distinguishing features of the Spiral Model. First, it is a cyclic process that aims to reduce risk. Second, it has anchor point milestones to ensure stakeholder satisfaction and to accommodate changes (Hansen, 2001).

Essentials of the Spiral Model

Spiral Essential 1: Determination of Key Artifacts (Operational Concept, Requirements, Plans, Design, and Code)

For a successful implementation of the Spiral Model, it is important to define certain artifacts that are not necessarily sequential and concurrent. Important artifacts include Operational Concept, the System and Software Requirements, Plan and Design, and the Code components including cost, algorithms, prototypes, reused components, and success-critical components. Ignoring these important artifacts could sabotage the project from the very beginning limiting the chances of success (Hansen, 2001).

Artifacts can vary greatly depending on the project requirements. Changes can be in areas such as the technology used or the strategies employed for the development process. Projects with low technological aspects are likely to be requirements intensive whereas projects with high technological aspects are likely to be code intensive. However, this does not change the number of cycles in the development process. Omission of any of these key artifacts could result in big losses for the delivery team as well as the rest of the stakeholders (Hansen, 2001).

Spiral Essential 2: Each Cycle Includes Objectives, Constraints, Alternatives, Risks, Reviews, and Commitment to Proceed

As with the other essential aspects of spiral, this one involves key areas of the project. The important activities included in this spiral essential area are identification of critical stakeholders' objectives and constraints, evaluation and elaboration of the project process, alternatives for achieving the objectives, identification of risks and mitigation strategies, and stakeholders' review and commitment to proceed with the defined plan. Ignoring any of these important aspects, again, could result in premature termination or failure of the project (Hansen, 2001). There are no predefined generic techniques to mitigate risk. Once risk identification and evaluation are complete, the project manager and the team along with the stakeholders come up with strategies they feel are appropriate to mitigate the identified risks (Hansen, 2001).

Spiral Essential 3: Level of Effort Driven by Risk Considerations

Spiral essential 3 is used to answer tough questions as to how many prototypes will be needed, how much engineering will be required, and how much testing and configuration management is needed for each phase with respect to the risks involved. Here, all possible risks involved are identified and evaluated in terms of affordable losses. Risk cannot be eliminated completely. However, what the development team aims for is risk minimization and strategies to combat risk should it present itself.

The organization generally chooses one particular method for risk assessment and management after all possible risks have been identified. Risk identification also helps identify how much testing needs to be done. In a case where risks are high, the team would usually have to do more testing comparative to lower risk projects. Ignoring any potential risk could severely cripple a project should it occur, but there is never enough time to handle all risks so this is a critical planning element. Natural disasters are also considered since they can have an impact on the development process.

Spiral Essential 4: Degree of Detail Driven by Risk Considerations

The fourth essential details the results of the efforts in the third spiral essential. The risk consideration identified earlier is addressed by artifacts here. This means that not every requirement of the software can be predefined and static. It should be flexible enough to incorporated changes by the client.

For example, a screen layout, if predefined, cannot incorporate changes later if the client wishes to. This could jeopardize the usability of the screen layout and hence create problems for the user. On the other hand, some requirements of the software should be static and predefined such as a security interface. No changes should be allowed once it has been initially finalized and implemented. The software requirements at the client's end and at the development team's end should always match to avoid complications. This can all be summed up in the two simple statements below (Hansen, 2001).

- If it is risky to NOT specify, then specify.
- If it is risky to specify, then do NOT specify.

Spiral Essential 5: Use Anchor Point Milestones LCO, LCA, and IOC

One feature lacking from the initial Spiral Model was that there were no properly defined milestones to serve as progress checking points. This made monitoring the development progress quite difficult. It was later modified to include three important milestones that are listed below (Hansen, 2001).

- **LCO - Life Cycle Objectives:** Describes what the system intends to accomplish.
- **LCA - Life Cycle Architecture:** Describes what the system's structure will be like.
- **IOC - Initial Operating Capability:** The initial fully functional release.

The main focus of LCO is to ensure that at least one of the proposed architectural designs is aligned with business values. The LCA review, on the other hand, focuses on committing to a single design that will be used throughout the project. By now the team should have either eliminated all possible risks or should have developed a risk management plan. IOC is particularly important since this is ultimately the pass/fail criteria for the team's work. If the stakeholders are satisfied with the initial functional release, then future releases are expected to be satisfactory as well (Hansen, 2001).

Each milestone is essentially a commitment of the stakeholders to support the process. At LCO the stakeholders agree to support the architecture, at LCA the stakeholders agree to support initial deployment, and at IOC they agree to support operations. The anchor point

milestones together help avoid unrealistic requirements, cost overruns, scope creep, and useless system functionalities (Hansen, 2001).

Spiral Essential 6: Emphasis on System and Life Cycle Activities and Artifacts

Spiral Essential 6 emphasizes that in software intensive systems that it is equally important to focus on system and life cycle concerns as well as software construction aspects. This means it is not only important that the software meets all requirements, but also that it should be adaptable enough to function in different environments, meet performance criteria, satisfy customers, and enhance business value. Simply writing code will not do the job. It is crucial for the team members to work together in harmony to achieve all these objectives successfully (Hansen, 2001). The Spiral Model's use of risk management makes it more efficient for the development team to produce an appropriate combination of hardware and software for the end solution (Hansen, 2001).

These six attributes must be incorporated in the use of the Spiral Model for it to be successful and produce the desired results. If any one of them are omitted, then it is no longer a Spiral Model implementation, but merely an incremental or cyclic effort. Spiral development works effectively with the use of evolutionary information systems. It helps with the combination of hardware and software in such a way that it minimizes risks and eventually satisfies the client (Hansen, 2001).

Spiral Model Stages

The Spiral Model is based on the concept of continuously improving releases by looking at the mistakes or errors of the previous increment. Each incremental release is improved and promises to be more effective. The phases of the Spiral Model are not much different than the Waterfall Model with the exception of a few additions. The stages are discussed below (Shah, 2008).

- Planning.
- Risk Analysis.
- Engineering.
- Customer Evaluation.

Planning: Requirements are gathered and processed at this stage and the scope and concept of the project is defined and explained to the development team. All objectives, challenges, alternatives, and constraints associated with the project are identified at this time. This time is critical. It is essential to spend proper time planning the project since this is the foundation of the whole effort and anything overlooked could jeopardize the success of the project (Shah, 2008).

Risk Analysis: In this stage all potential risks that are likely to occur are identified and analyzed carefully. This is the most important stage of the Spiral Model because it evaluates the possibilities of success of the project. At this stage the potential risks are identified and solutions are derived to combat the risks. Additionally, alternative options are created to ensure the project development process is not interrupted by any risk (Shah, 2008).

Engineering: Once all risks have been identified the next step is to actually execute the plan, which is otherwise known as the development and implementation phase. This, of course, according to the Spiral Model, is completed through the use of prototypes developed during each quadrant of the model diagram (See Figure 16.1). In some cases, simulation is also used (Shah, 2008).

Customer Evaluation: Each incremental release is evaluated by the customer. The hardware and software compatibility is tested and the customer provides the team with crucial feedback. Any changes or modifications asked for by the client are made and delivered in the next release. This helps to consistently improve project value. The Spiral Model is one of the most realistic approaches to software delivery. It makes handling large projects easy and customer feedback enhances the project's alignment with business value.

Win-Win Spiral Model

The Win-Win Spiral Model is a modified version of Boehm's Spiral Model. The Spiral Model is comprised of a framework where requirements are explicitly stated by the client and the delivery team follows them. However, this is rarely the case. Usually, the client and the development team enter into a process of negotiation. At that point, the customer is asked to provide realistic requirements that are in accordance with the time, budget, and resource constraints (singh, 2010).

The delivery team and the client should, of course, strive for a win-win situation. For the customer, this is when the software developed is able to fulfil all their requirements,

specifications, and satisfies their needs completely. For the development team, it becomes a win-win situation if the software satisfies the customer's needs and is built within time, schedule, and budget constraints (singh, 2010).

The Win-Win Spiral Model basically defines a set of activities that can be used by the development team while negotiating the terms of the contract. These activities are listed below (singh, 2010).

- Identification of the stakeholders for the system and subsystem.
- Determining the "win" situation for the stakeholders.
- Negotiate the stakeholders' win condition and incorporate it with the team's win condition to increase the probability of success.

Successful completion of these activities results in achieving the win-win situation for the team as well as for the client. This avoids undesirable conflicts and problems with the client and enhances the communication process (singh, 2010).

The Win-Win Spiral Model also includes the three anchor milestones mentioned before to further strengthen the probability of success. These milestones are still called Anchor Points. Like before, they help in completing cycles and provide milestones to evaluate the success before deploying the release to the client. In the Win-Win Spiral Model the anchor milestones are still LCO (Life Cycle Objectives), LCA (Life Cycle Architecture), and IOC (Initial Operational Capability) (singh, 2010).

In the end, the Win-Win Spiral Model is similar to the Spiral Model proposed by Boehm earlier. The only difference is that it emphasizes the importance of negotiation (singh, 2010). It is an extension of the original Spiral Model and includes the same phases of development. However, the model is faster and easy to use when involving all stakeholders. Additionally, it is also cheap and reduces maintenance cost.

Spiral Model vs. Waterfall Model

Software delivery is not an easy task. Over the years there has been many software development methodologies created to achieve success but none of them produced 100% of the required results. As mentioned before, the Waterfall Model was among the first ones to be

used and then the Spiral Model was designed later. Both of these models are very popular in the field of software development.

The comparison between the two models is an ongoing debate among developers. They differ from each other in many ways. They have different aspects and developmental strategies that make it appropriate for some software while inappropriate for others. The model selection will depend entirely on the type of software being developed. Understanding the differences between the two models will help the developer figure out which model to employ and how. We will discuss these models briefly while comparing their advantages and disadvantages below (Satalkar, 2011).

Waterfall Model

The Waterfall Model has also been called the Linear and Sequential Model. As the name suggests, the flow of activities is either linear or sequential. The next activity cannot be started unless the preceding activity has been completed successfully. Once the activity has been completed and the process moves on to the next stage, they cannot go back. The model doesn't support the concept of changes in the activities once they have been completed. The stages in the Waterfall Model are below (Satalkar, 2011).

- Requirements Gathering Phase.
- Analysis Phase.
- Design Phase.
- Coding Phase.
- Testing and Debugging Phase.
- Acceptance.

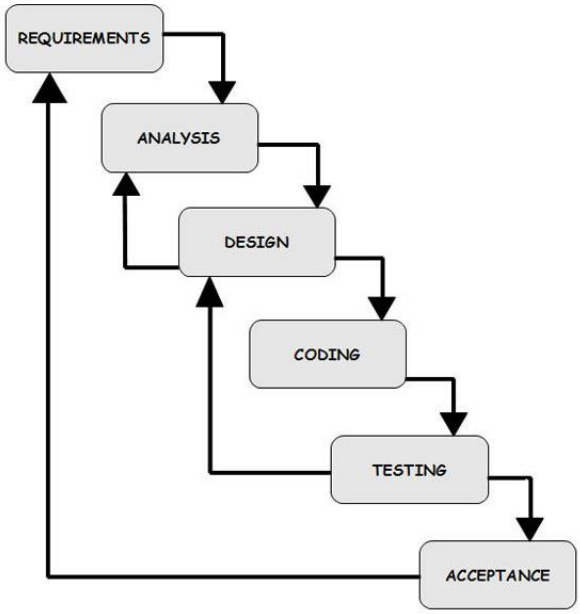


Figure 16.2 The Waterfall Model

Advantages of the Waterfall Model

One of the main advantages of the Waterfall Model is its approach to planning. Each activity is planned upfront before it is about to start. The process with the Waterfall Model is disciplined as changes are not encouraged and predefined rules are followed. Additionally, confusion and conflicts are avoided since operations only occur in one phase at a time. Comparatively, it is hard to identify the start and finish with other software development models because they generally encourage working on more than one phase at a time. Furthermore, with the Waterfall Model, resources are typically not wasted as they are allocated to their respective phases and are usually stable (Satalkar, 2011).

Disadvantages of the Waterfall Model

One of the main disadvantages of the Waterfall Model is its inability to incorporate changes. Once the requirements have been frozen they cannot be changed no matter what. This at times can frustrate the customer as they might have forgotten a few important requirements in the initial planning stage. Additionally, the model does not support returning to a previous phase once it has been completed and finalized. Furthermore, as with many other life cycles, if the project schedule is too long potential obsolesce of hardware and software can risk the success of the project (Satalkar, 2011).

Spiral Model

Due to the shortcomings of the Waterfall Model the Spiral Model was introduced. As the name suggests, project activities are carried out in a graphical spiral (Figure 16.1). The Spiral Model includes the phases listed below, as detailed earlier in this book (Satalkar, 2011).

- Planning.
- Risk Analysis.
- Engineering.
- Customer Evaluation.

Strengths of the Spiral Model

Basically, the disadvantages of the Waterfall Model are the advantages of the Spiral Model. It is a realistic model based on rational approaches used for delivering larger and more complex software systems. A systematic approach is used combined with incremental development. This reduces chances of bug occurrences in the software and satisfies the customer. Additionally, changes are encouraged and can be incorporated at any process stage (Satalkar, 2011).

Weaknesses of the Spiral Model

It is of utmost importance to have a risk manager on the development team. Failure to identify risks can result in the complete failure of the overall project. Furthermore, unidentified risks can result in the client having to spend unplanned time with the development team to rectify issues. This will create schedule slips and increase costs. Additionally, planning of the project is another area that requires a great deal of the client's time as they work with the development team. This is a process that can also lead to unnecessary involvement of the client making it hard for the development team to work effectively (Satalkar, 2011).

Differences between the Waterfall and Spiral Models

The Spiral Model dictates that the customer's presence is of utmost importance and they are made aware of all the activities in the delivery process while with the Waterfall

Model the customer is usually not involved. This often leads to customer dissatisfaction as the software may not be developed according to the customer's requirements. Additionally, the Spiral Model allows the customer to speak on about every aspect of the project, which greatly increases the chances of success (Satalkar, 2011).

In the Waterfall Model, once a phase has been completed there is no going back. This can create problems and block production. Furthermore, it can be especially challenging during the coding phase where the original design may have seemed straight forward on paper, but the actual implementation becomes problematic. This makes it imperative that the code is correct the first time. However, with the Spiral Model, the software is delivered in increments that give the team an opportunity to resolve any bugs. The activities and architecture of the model are designed in such a way to support changes (Satalkar, 2011).

In the Spiral Model, the development team can revisit any of the previous phases whenever they want. Additionally, they can review and revise as needed. This process option greatly assists in defect tracking and resolution. Unfortunately, the same isn't possible with the Waterfall Model (Satalkar, 2011).

That said, often people confuse the two models due to their similarities. The Spiral Model is perceived to be for complex projects as it involves a lot of development cycles with the least amount of paper work. The process is complex and no record is maintained. Whereas the Waterfall Model is a more systematic project management approach where every detail is documented, making it easier to understand and review when needed (Satalkar, 2011).

Discussing each model individually and then collectively shows that each model has their respective pros and cons. Which model to use, depends entirely on the requirements of the project. The project requirements should be evaluated thoroughly before opting for any approach. The size of the project and the time allocated also plays an important role along with the resources available (Satalkar, 2011).

Advantages of the Spiral Model

The Spiral Model has the advantages listed below (G, 2010).

- Significant risk reduction through continuous and repeated development.

- The Spiral Model is adaptable. It can accommodate changes at any stage of the process.
- Since project prototypes are developed in increments the client has more control of cost estimation.
- As the project progresses towards completion customer satisfaction generally tends to grow as they are involved in the process from beginning till end.

Disadvantages of the Spiral Model

The Spiral Model has the disadvantages listed below (G, 2010).

- The Spiral Model is best suited for larger projects where there are complex requirements involved with higher costs, thus smaller projects can be troublesome.
- Depending on the skills and experience of the development team, the risk evaluation process could be completed inadequately leading to overlooked risks and project issues.
- Evaluating every risk can increase project costs and exceed budget if not controlled.
- Requires extensive client engagement in the process.

The Spiral Model has been one of the most influential models in use, yielding more successful results compared to other SDLC (System Development Life Cycle) models. It gained success in a very short period of time especially when it came to developing large complex projects. Being an incremental approach to development that includes risk assessment and management capabilities makes it a viable approach. With a little insight into the Spiral Model any organization can adapt it. Even if the organization is not ready to change their overall development process, they can still implement the Spiral Model for a few projects.

Chapter 17

Synchronize and Stabilize

Management of large scale products is a challenging task. It's not only costly, but is also complicated. Additionally, the coordination between the team members is often hectic and confusing. Furthermore, companies and organizations working with mega projects face the difficulty of complicated procedures and huge expenditures.

Usually the modifications or increments are completed at the end of the development process. Additionally, it is not uncommon for several life cycle models to be combined to form a hybrid methodology to plan and then maintain a development procedure. All of these drawbacks demand more flexible design practices.

The challenges faced by contemporary project management teams are that the technology is evolving so fast that they can hardly keep pace with it. Software development is becoming more challenging because of the diverse needs of end users that have to be managed. Furthermore, the software applications developed today are far more multifaceted. In order to manage different aspects of software development, there is a need for one solid methodology that can cater to all these facets and produce a coherent result in the minimum delivery time possible.

History of Synchronize and Stabilize

In 1998, Yoshioka, Suzuki, and Katayama formed a method of writing programs by the name of Incremental Software Development using Data Reification (ISDR) (Rouse, 2005) (Janssen, Synchronize and Stabilize). The whole premise of this method was to write code with specifications being generated later (Sathiparsad, 2003). The idea was that the code could be refined as the process proceeds. This model became the foundation for what is now the Synchronize and Stabilize methodology.

The Synchronize and Stabilize approach was developed by David Yoffie of Harvard University and Michael Cusumano of Massachusetts Institute of Technology (Clair, 2005). Additional names that have been given to this methodology are Milestone, Daily Build, Zero Defect Process, and Nightly Build (Singh, 2012) (Cusumano & Selby, 1998). Another interesting name given for this methodology is FWITW aka Fiddle With It Till it Works.

Using Synchronize and Stabilize, the members of a delivery team work on modular blocks of code while regularly synchronizing their code with other team members throughout the development process. The members of the team also stabilize and debug continuously throughout the development process. In the past, the software development methodologies focused their strengths on catering to the needs of large organizations and adjusting to production of large products. With changing times, the focus is moving to small and medium scale organizations that operate under constraints of restricted resources and budgets (Sathiparsad, 2003). The degree of competition and uncertainty is also higher, which calls for greater innovation in the development process.

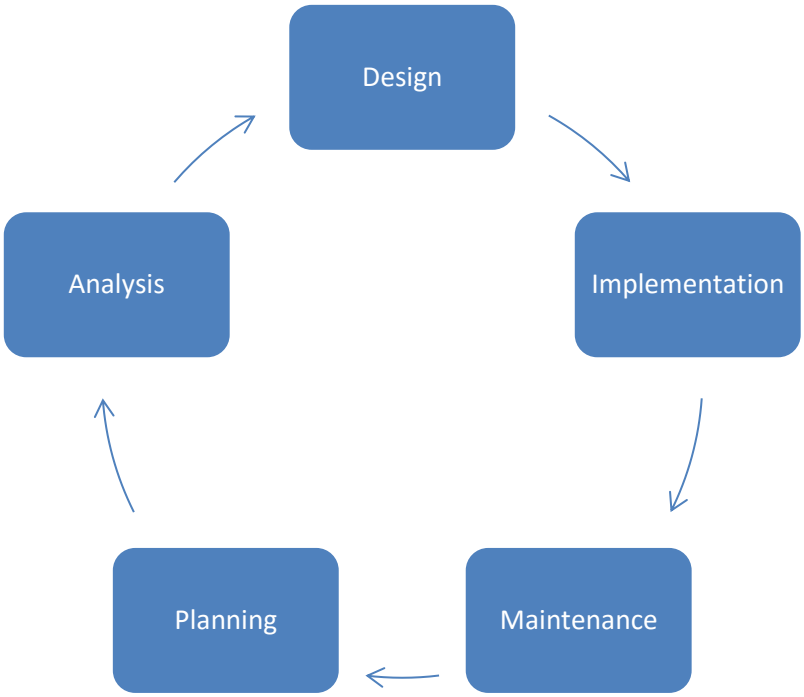


Figure 17.1 Synchronize and Stabilize

Figure 17.1 above shows the Synchronize and Stabilize process. Here you can see how the five steps of the development cycle continue every day until the final version of the product is completed and ready for release. All steps are performed simultaneously for a rapid yet coherent end result.

A historic application of Synchronize and Stabilize was studied as a comparison between the development of Internet Explorer (by Microsoft) and that of Netscape Communicator (by Netscape Communications). A comparative study of these software development projects indicated that a rough draft of this methodology was used and code was completed nightly and then disintegrated, but procedural efforts were combined through

stabilization. With this research conducted by Yoffie and Cusumano, Synchronize and Stabilize became famous.

What is Synchronize and Stabilize?

Synchronize and Stabilize is a software project life cycle that is based on the premise that a large project can be broken down into smaller parts. These smaller parts are then coded by individuals or teams of individuals. Later the parts are combined for project completion. In this way the overall project is a cost-effective and time-effective success.

A project may be divided into as many milestones as required. Typically, there is an alpha, beta, and final release. However, every organization can adapt the release schedule to align with their distinct requirements.

Alpha release is normally used for testing within the organization. Beta releases are used to assess the public's response to the limited version of the final release. The final release, or gold master, is the final developed product. This methodology offers a number of advantages over linear development methodologies. We will discuss the advantages of this SDLC methodology later on.

Synchronize and Stabilize is not altogether a new creation. Like many life cycles meant to provide a better process, it is built on the foundation of the Waterfall methodology. It is just that the steps that are processed methodically in Waterfall have been used in a more creative fashion in this methodology. This transformation of implementing the steps brings about enhancement in quality, time savings, cost savings, and other benefits.

Most products are designed linearly such that development can take place in parts and the components can be combined later (Cusumano & Selby, 1998). Continuous testing of the product being developed is executed to ensure that operations perform predictably and as planned. In addition to the wide acceptance of Synchronize and Stabilize in project management and software development, the method is also enjoying adoption in engineering and related disciplines. Anything and everything that can be completed by using an incremental approach can be accomplished with this methodology.

The Two Parts

As evident by the name of this methodology, there are two key parts. They are, of course synchronize and stabilize (Singh, 2012). The "Synchronize" portion of the name is

derived from the different teams or individuals working on different segments of a larger project that are later combined so that the overall project seems like a homogeneous effort. This may involve combining various code modules or even outputs from various processes.

The "Stabilize" segment of the methodology references the process after merger of the sub-components that makes sure that there are no inconsistencies in the final merged product. It may include debugging or streamlining to an agreed format (Singh, 2012). Thus when both the synchronization and stabilization operations are performed at the end of the day (or at the end of the interval duration), the product should be cohesive and ready for further development or deployment.

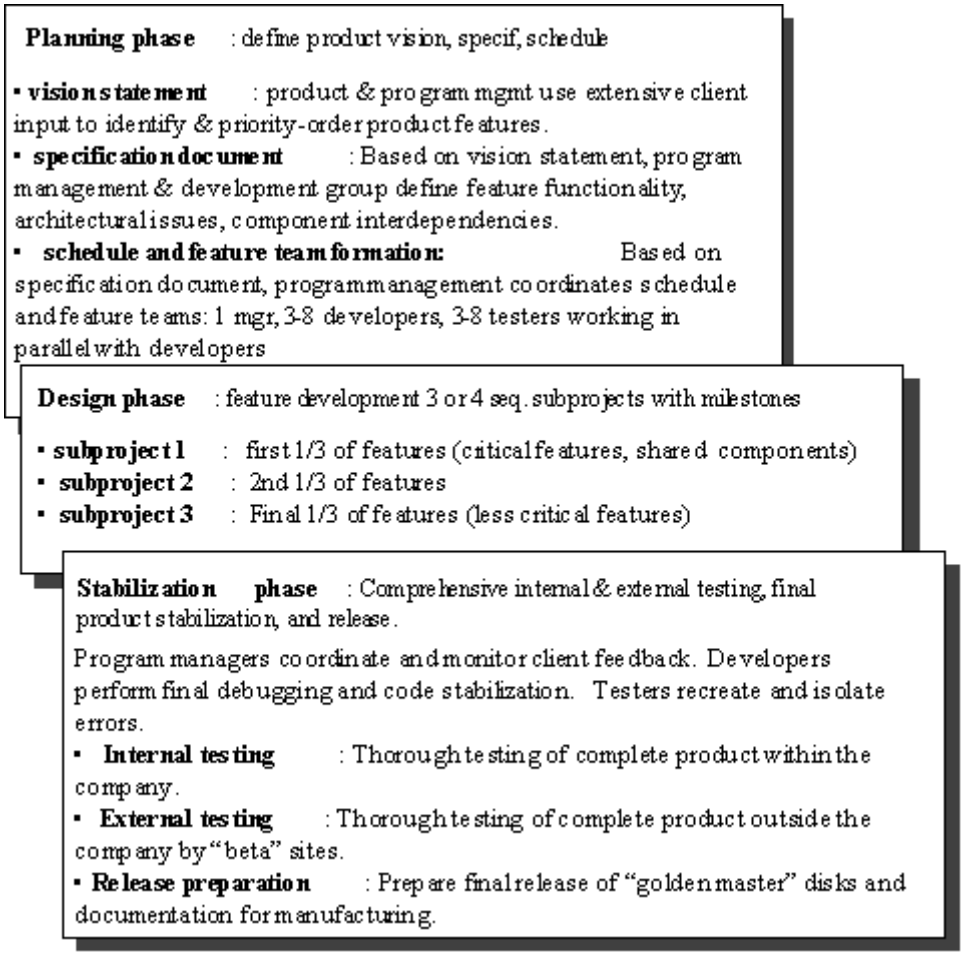


Figure 17.2 Various Components of Synchronize and Stabilize

Microsoft's Implementation

Microsoft is a leading name in the computer industry that everyone who has ever touched a keyboard has heard. They evaluated various development methodologies before they finally settled on Synchronize and Stabilize. Synchronize and Stabilize gained

acceptance at Microsoft for its effectiveness in delivering large scale projects (McConnell, Daily Build and Smoke Test, 1996).

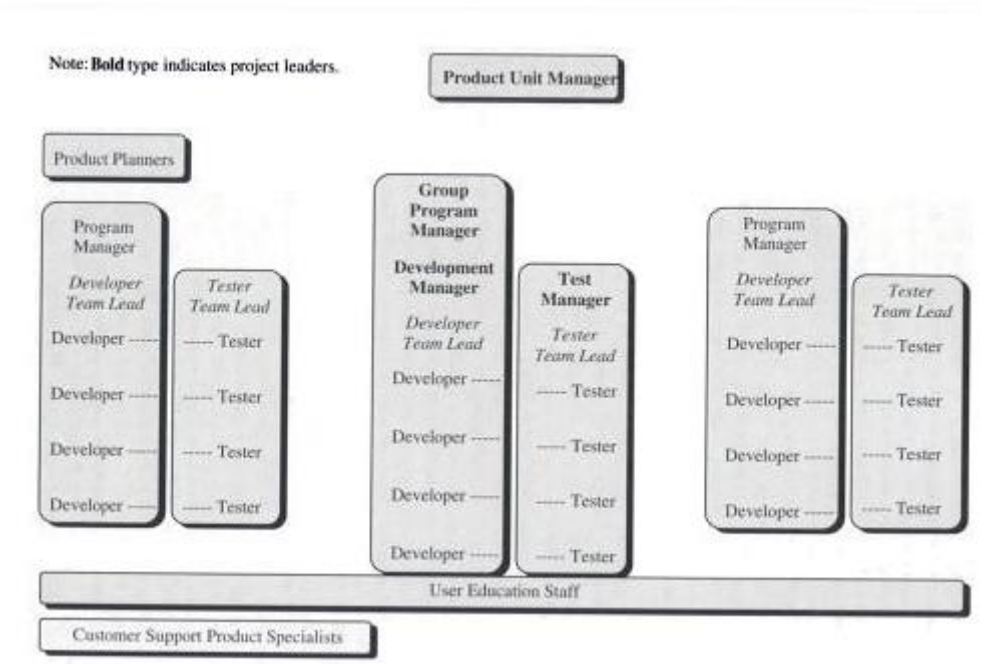


Figure 17.3 Synchronize and Stabilize Project Organization at Microsoft

Figure 17.3 demonstrates how product development is executed at Microsoft using Synchronize and Stabilize. In the figure, it can be seen how large teams are broken into smaller teams. Later, their work will be rolled up into one iteration of the product for stabilization.

The objective here is to break down any given project into small manageable parts and ultimately start the development with baby steps. If there are any problems, the delivery team can't move ahead until they are fixed. This incremental approach ensures that as the team moves forward, their work product is free of defects. Thusly, milestones are completed and results are integrated as work moves forward. However, Microsoft's implementation of Synchronize and Stabilize has a bit more aggressive focus in that they strive to have no project exceed more than a few months, which would lead to valuable resources being wasted (Cusumano & Selby, 1998).

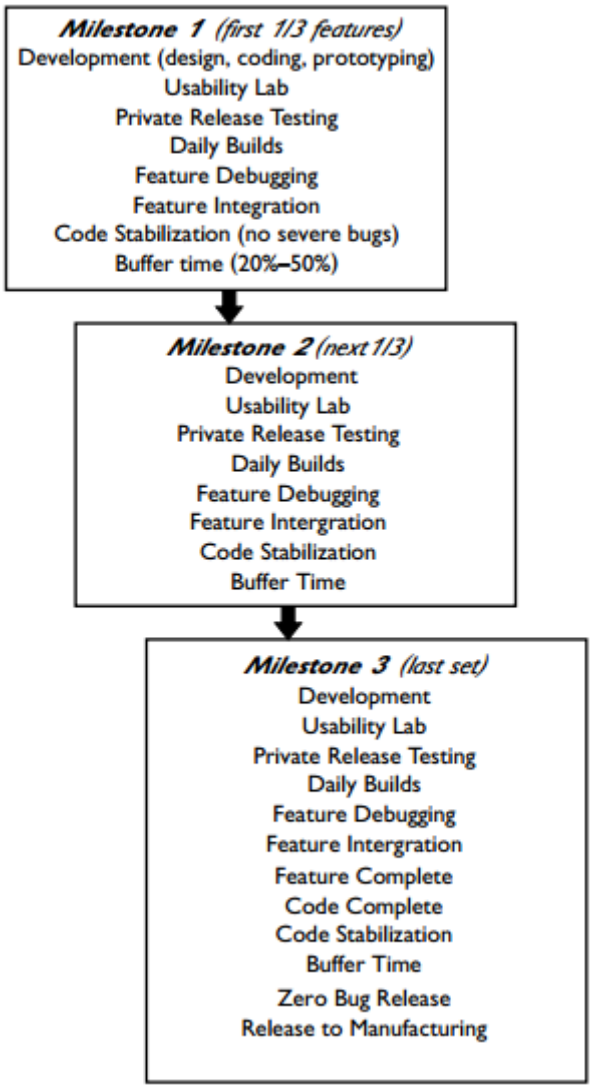


Figure 17.4 Synchronize and Stabilize Milestone Examples

With Microsoft's aggressive schedule, things have to move much quicker. Developers at Microsoft need to combine their diverse code efforts at the end of each day so that work progress can be assessed (Singh, 2012). If there are any discrepancies, they are fixed before proceeding with the next day's work. Microsoft has used Synchronize and Stabilize in renowned products like Internet Explorer, Microsoft Office, Publisher, Windows 95, Windows 98, Windows NT, and numerous other products (Malik & Palencia, 1999). More than 3000 developers and testers worked together to make Windows 2000 a success using the Synchronize and Stabilize methodology. Furthermore, more than 30 million lines of code were written while they were still building their foundation on Windows NT 4.0.

With huge successes like these under their belts, Microsoft continues to improve upon the implementation of this methodology and keeps applying it in one way or another. The

success of Synchronize and Stabilize at Microsoft gave innumerable medium and large organizations the confidence they needed to implement this methodology in their own organizations. Microsoft's success with their implementation, however, does not guarantee that other organizations will be as successful. Through the years Microsoft has mastered its use of Synchronize and Stabilize and continues to improve upon its practices to the point that today they have a significant advantage over competitors (Schach, 2007).

Figure 17.5 Synchronize and Stabilize Development Process as used by Microsoft

Synchronize and Stabilize using Components

Synchronize and Stabilize using Components is a concept that implies use of third party software components that aid in the deployment process (Sathiparsad, 2003). This speeds up the whole development process and makes the job of the delivery team easier. They can easily bypass the steps that are otherwise satisfied by third party code. This practice often has the effect of reducing costs while augmenting the project with additional expertise since the design team can select from the best options available.

Another option for implementing Synchronize and Stabilize using Components is to assign another team within the organization to the development of components while the main team is engaged with larger sections of code (Sathiparsad, 2003). In this way, both the teams will have their work completed sooner and later harmonize their merged work outputs. If component selection is performed carefully, considerable time can be saved and also the components can be reused at any time. However, when developing with the use of components, it is vital to keep close watch on hidden costs and compatibility issues so that problems will not arise later.

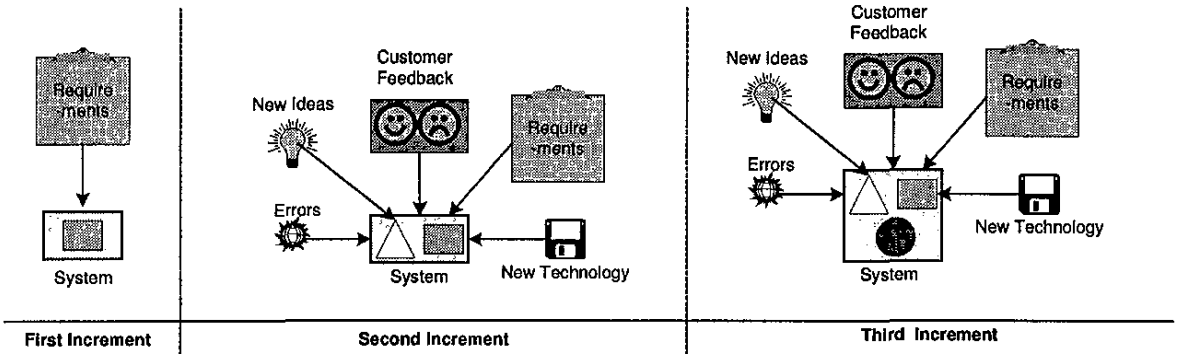


Figure 17.6 Incremental Process of Synchronize and Stabilize

Testing Team and Supervisor

In large scale projects or huge organizations, there is a supervisor, program manager, or testing team who is entrusted with the task of coordinating the efforts of the autonomous teams who are all working on the project simultaneously. If it is not a burden to the project budget, it is extremely advantageous to assign someone to manage and oversee the efforts of these diverse teams. The testing team will synchronize the work output from the sub-teams and combine them using a standard stabilization approach.

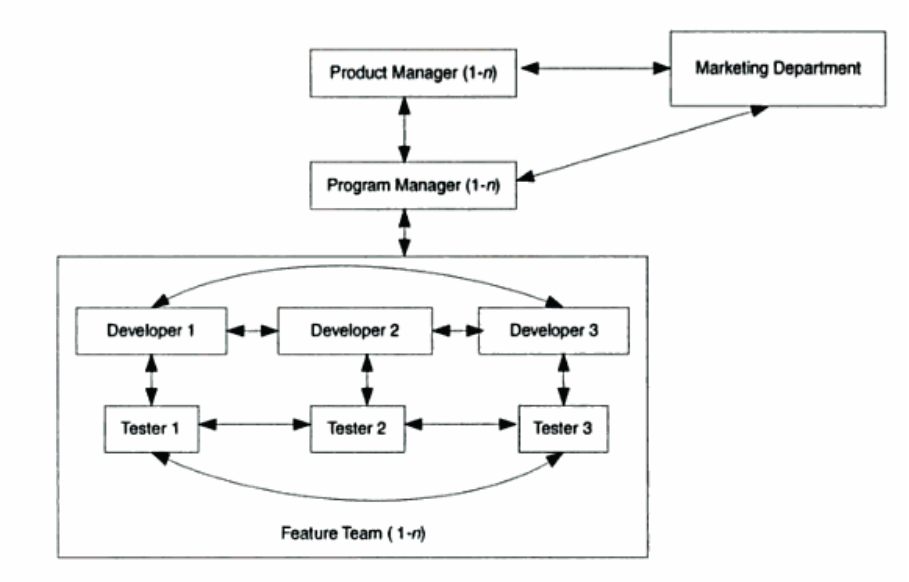


Figure 17.7 Synchronize and Stabilize Team Break Down

Figure 17.7 shows a rough framework for implementation of work division between developers, testers, program manager, and other senior staff members. If no separate testing team, program manager, or supervisor is hired, it is important to delegate this responsibility to a person or persons so that someone bears the responsibility and has the authority to administer the whole process and make sure the project is a success. However, in some organizations, the person assigned this responsibility has to take on the additional effort while maintaining their current technical role on the team. This divides their attention and keeps them from focusing 100% on their technical tasks or their management duties (Webster). For this reason it is far better to have a dedicated person or team whose sole responsibility is the success of the project.

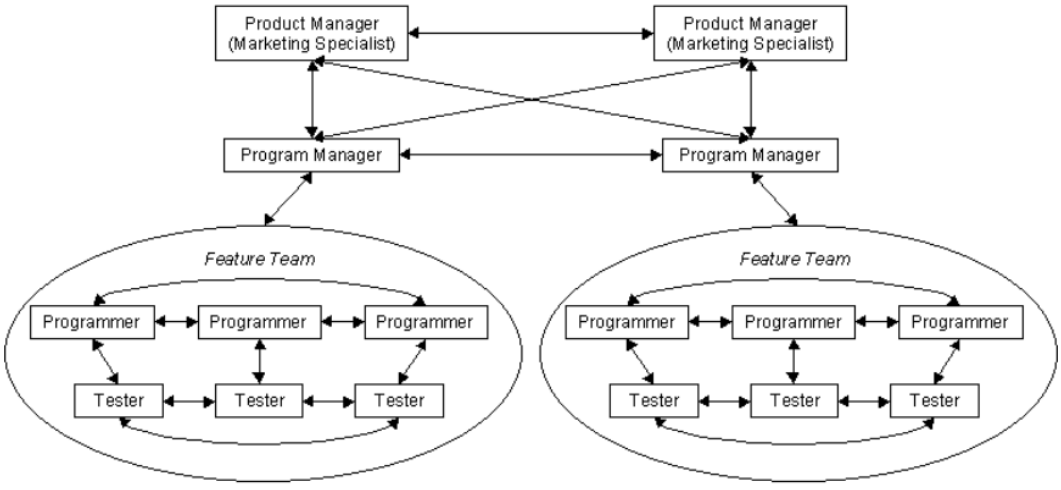


Figure 17.8 Synchronize and Stabilize Team Organization

Advantages of Synchronize and Stabilize

As with the other chapters and models in this book, we will now discuss advantages and disadvantages of Synchronize and Stabilize. Not surprisingly, there are significant benefits to using this methodology for project management and software delivery. Below, we discuss those advantages in greater detail.

First, the Synchronize and Stabilize model is another method that provides an overall approach that makes delivery of large scale software systems more efficient even though it is not designed solely for large projects. It enables sub-teams to work in parallel on many individual code segments and later roll-up there combined efforts into a final product. Each team completes a development cycle comprised of integration, testing, and debugging (Malik & Palencia, 1999). This is completed by way of programmers, as individuals and as members of parallel teams, frequently synchronizing and periodically stabilizing or debugging their code regularly throughout the development process.

Second, Synchronize and Stabilize is generally more flexible than older approaches like Waterfall, Spiral, and Agile. This has driven great acceptance of the model when the software product is innovative. In an internet age, innovation happens faster and faster every day. In fact, it is often measured on an exponential scale.

This flexibility is contrary to more rigid models that are often used for military and medical systems. It allows for modifications whenever and wherever needed during the project. However, the significance of this model is the balance between structure and flexibility in software product development. This balance is due to the other key component of the model, which is periodic stabilization. All of this allows for design features to be easily added or removed from various versions of the product until the right set of features are achieved as desired.

In short, the Synchronize and Stabilize methodology is ideal for today's fast-paced and continuously changing markets where complex enterprise products are developed within short time frames. It caters to market changes easily. Additionally, it offers an excellent mechanism for coordination by allowing disintegration of a large team into smaller interdependent teams that work individually.

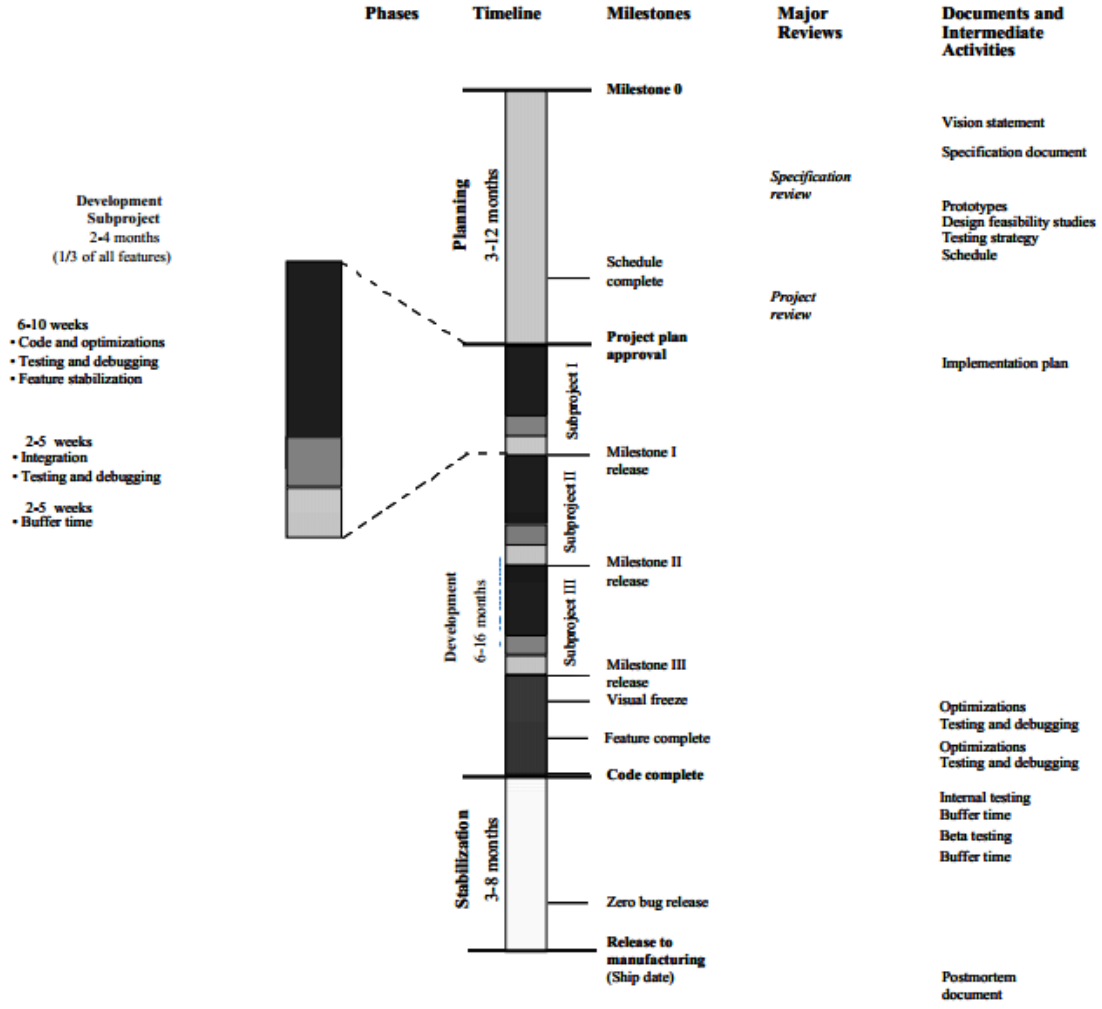


Figure 17.9 Synchronize and Stabilize Life Cycle Model for Large Scale Software Systems

Third, when using Synchronize and Stabilize for product delivery, each team may encounter different challenges per coding increment. For this reason, people with specialized skills can be hired for each part of the project. Thus, the best results are clearly fostered by employing the appropriate human resources for each aspect of the project.

Fourth, it is generally felt by practitioners that debugging every day, as is done in the stabilize portion of the product delivery, or at the end of every module, is easier. This is because the testers and developers can easily point out where the problems might be. On the other hand, finding bugs in the system as a whole can be a more complicated task.

Fifth, another critical aspect of any product development initiative is time to market. Time to market is always a critical factor for any company engaging in new product introduction. This is quite simply because the first to market stands a better chance of market dominance and higher return on investment. With an incremental release strategy for market

introduction, an organization has its best chance of ensuring that they receive their intended share of the market (Malik & Palencia, 1999). This is another area where Synchronize and Stabilize shines because the approach reduces the time to market as compared to one grand release at the end of the development process.

Sixth, often with software products compatibility can be an issue. This can be anything from operating system compatibility to old versus new software or even, of course, multiple software products working together in concert. However, these issues are also addressed with the use of Synchronize and Stabilize for product development.

Seventh, engineering has an artistic component to it and for that reason many engineers like to work unrestricted or at least with few restrictions. With Synchronize and Stabilize, individual members of any given team are given a bit of autonomy as long as they continue to code within the architectural guidelines of the project. This limited creativity can assist in engaging team members with the project.

Eighth, tracking to the triple constraints of project management (time, scope, and resources) is critical with any project and it is no different using Synchronize and Stabilize. However, project feasibility and budget can be determined after each milestone has been achieved with Synchronize and Stabilize thereby maintaining successful tracking within the project. This allows the project manager or supervisor to take any remedial actions needed to maintain a successful path for the project.

Not everyone embraces Synchronize and Stabilize though. In fact, some proponents of other project management life cycles argue that Synchronize and Stabilize is a variation of the hacker mentality of a past era. However, the methodology saves itself from this slant with the daily synchronization that takes place during the project.

Ninth, there is also an additional advantage to the incremental release approach provided by Synchronize and Stabilize that should be mentioned. Following the incremental build approach of the methodology, the development team can also make use of the valuable feedback that comes their way via end users and customers. In this way, customer satisfaction can be increased through requirements changes based on end user opinion and adapted to the final release of the product.

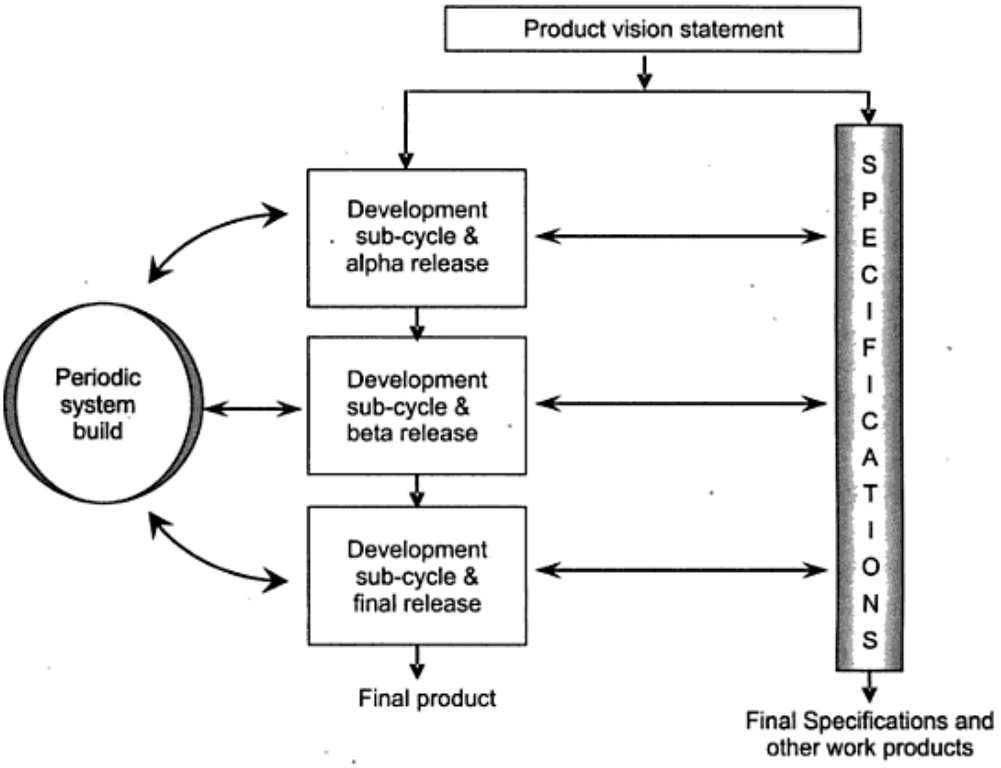


Figure 17.10 Synchronize and Stabilize Model

Tenth, overall risk is also reduced by employing Synchronize and Stabilize. This is because any risks that actually become an issue can generally be managed through the continuing development incremental release portion of the process (Sathiparsad, 2003).

Summary of Advantages of Synchronize and Stabilize

- It provides an overall approach that makes it easy to manage and develop large scale software systems even though it is not just for large projects.
- More flexible than older approaches like Waterfall, Spiral, and Agile.
- Shorter duration projects for quicker time to market.
- Team environment makes it easier to backfill missing resources.
- Resource skillsets can be augmented with additional resources as needed.
- Less software product compatibility issues.
- Team members are afforded a level of autonomy during design.
- Product improvements through Customer feedback.

- Risk reduction through incremental releases.

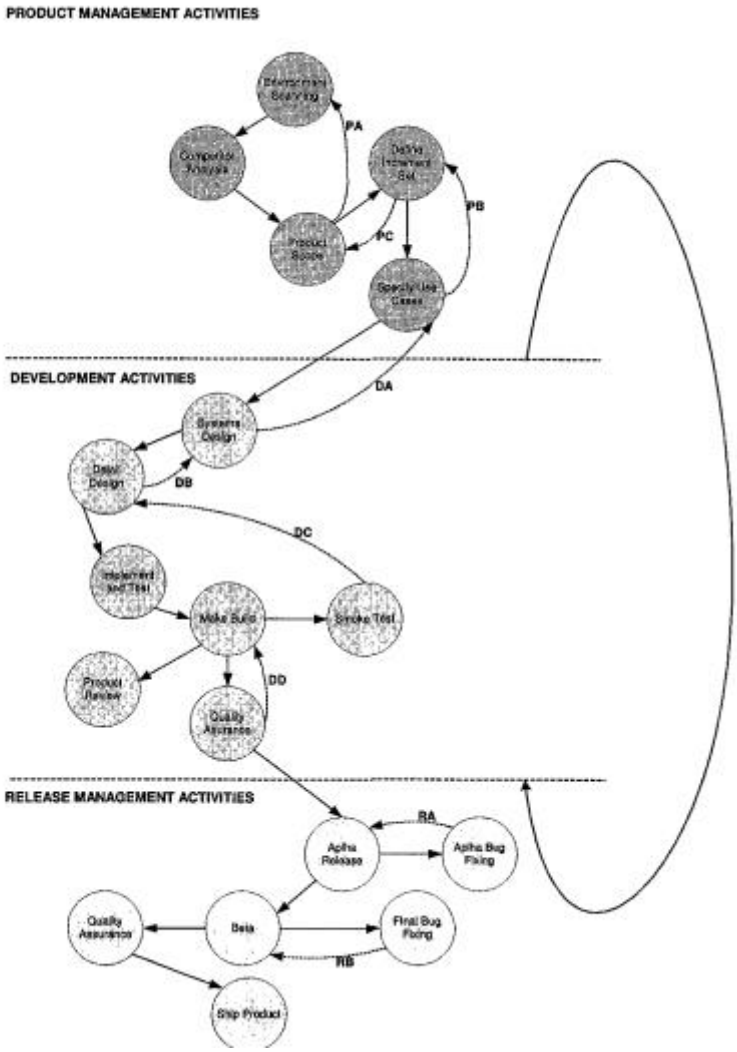


Figure 17.11 Typical Workflow Using Synchronize and Stabilize

Disadvantages of Synchronize and Stabilize

With the good there is always bad. Synchronize and Stabilize is no different. Below we discuss the disadvantages of Synchronize and Stabilize in greater detail.

First, there are the human resource requirements. This methodology requires an independent testing team that ensures the output from each individual or team is properly integrated (synchronized) and that there are no code defects (stabilize). This is not to say that other methodologies don't require testing, only that with Synchronize and Stabilize there needs to be a testing process just for the rollup effort.

Second, some people like documentation and some people don't. This will be a timeless argument. With Synchronize and Stabilize the specification for the project is not completed until the end of the project. This means that basically the team is only working with a base plan that can change at any time (Singh, 2012).

Third, segregated teams may have trouble working well together (Malik & Palencia, 1999). Not all teams are at ease working out of their comfort zone and coordinating with others. Thus considerable care should be sought to make sure that coordination among the teams is optimal.

Fourth, any time you work on sub-components of a project to be combined later there can be problems. It is the equivalent of starting a bridge on both sides of a river at the same time. Synchronize and Stabilize is no different. If the various teams working on different parts of the same project are not closely aligned, rollup of the sub-components can become difficult.

Fifth, while at one end of the spectrum Synchronize and Stabilize tends to reduce overall software complexity by breaking it down into small manageable tasks performed by individuals or teams independently, on the other end of the spectrum, this method tends to increase the complexity with the same organizational structure. This is ultimately because of the autonomy component mentioned above. All of the teams still need to come together as one mind on how to complete the overall product delivery effort. In short, the results of the different teams must be able to be rolled up in such a way that the final product is free of defects.

Sixth, as mentioned before time-to-market is an important part of product launch. No doubt Synchronize and Stabilize reduces the time-to-market with its multiple release approach. However, one drawback inherent with this methodology is that the repetitive releases ultimately tend to delay market entry point of the final product.

Seventh, the pressure to submit work daily does not necessarily translate into quality deliverables. It is possible that at the end of the day, a developer does not have something worthwhile to contribute, yet knows that he has to submit something to integrate with the efforts of others. This pressure may lead to inconsistent performance.

Summary of Disadvantages of Synchronize and Stabilize

- The method is testing intensive since each rollup must be tested.
- Lack of documentation during development since the specification is not complete until the end of the project.
- Team members may not like the sub-team structure because they have trouble coordinating with others.
- Rollup of sub-components can be difficult if the teams are not closely aligned.
- Getting the sub-teams to agree on the project approach may be difficult.
- Market entry of final product is delayed with an incremental release methodology.
- Daily code submittals can lead to low quality work due to pressures to perform.

Chapter 18

Reverse Engineering Development

Reverse Engineering is a process through which technological principles of a device, component, or a system can be discovered. Through proper and thorough evaluation and analysis of a system, its function, detailed operations, and the technological processes can be uncovered. To study a product thoroughly, the Reverse Engineering process can sometimes involve the disassembly of every functional part. At this point, it can be studied in detail. This disassembly helps to understand basic functionality and operation of the components and how they work when integrated as a system. Unlike many other life cycles in this book, this is one that can be used for many different products other than just software.

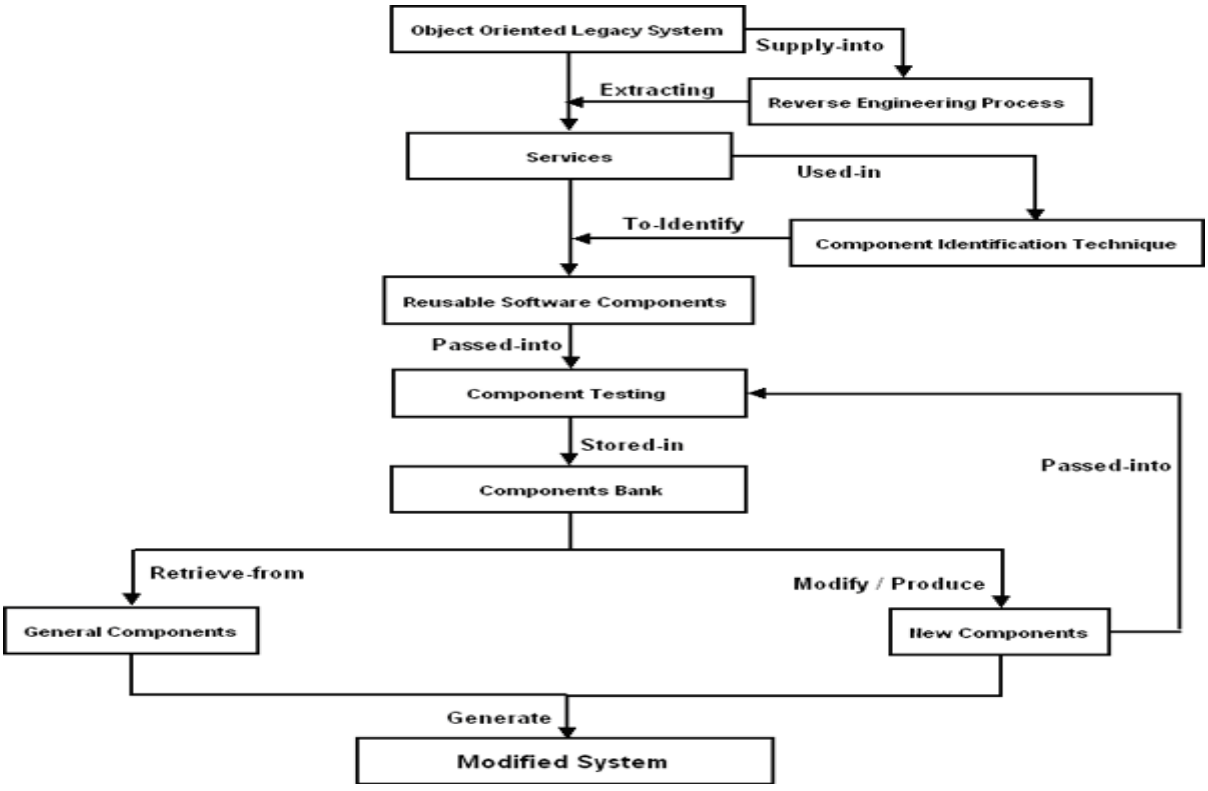


Figure 18.1 Example of Reverse Engineering Development

History of Reverse Engineering

The history of Reverse Engineering is much like that of the Prototype Model. From the very first completion of anything, probably including the wheel, someone was sitting around trying to figure out how it was done and how it could be better. One of the historic cases of Reverse Engineering was seen in the year 1987 when MAC OS System 4.1 was

reverse engineered in Bell Laboratories. While the original system was running on the Apple Macintosh SE, with the use of Reverse Engineering, it was able to run on Bell Laboratories RISC machines as well (Reverse Engineering, 2016). Another historic instance of reverse engineering that gained much attention in the 1980's was the reverse engineering of IBM PC's BIOS. In this case, IBM's PC BIOS was reverse engineered, which allowed for the growth of the IBM PC Compatible computer market (Schwartz, 2001).

Product Development Approaches

There are many who would not consider Reverse Engineering Development to be a project life cycle or a design approach, but merely a task in a greater process. There are even engineers that would argue against Reverse Engineering as a method of product development feeling that a better product could be made from the ground up without using a process like this. However, it is hard to ignore how products actually come to market in the real world. In fact, today, the methods for managing product design could be broadly classified into two different categories. They are simply called a "conventional approach" and a "non-conventional approach". (M. Sokovic, 2005)

Conventional Approach

As an example, a conventional approach to product development may begin with CAD/CAE/CAM techniques. It is quite common to start with geometric modeling techniques utilizing a CAD system. The geometric model could be represented as a wire frame or as surfaces or as a solid structure. Depending on proposed materials for the product, a computer model could even be subjected to material stresses as well as other tests.

Via conceptual modeling, the generated CAD information could be exported in a standard format (IGES points/STL binary, ASCII data, DXF polyline, VDA points or IGES/STL surfaces) and imported in the same data format to CAE systems. Afterward, numerical model simulations could be performed or the files could be imported to a CAM system that would allow for tooling selection and CNC programing for production of the product. Furthermore, in a system with a unique database, the design information could be shared between every application automatically without the need to transfer data manually (M. Sokovic, 2005).

Non-Conventional Approach

That said, the focus here is not conventional engineering and it only serves as a comparison to what is covered below. A conventional product development approach is not applicable when the goal is to reengineer or to simulate and optimize parts/molds/tools already existent without information in CAD data format. Consequently, it is necessary to apply techniques that allow for capturing the geometry of parts/molds/tools or prototypes, and to generate a conceptual numerical model that can been used in CAD and CAE. This process is usually referred to as Reverse Engineering. It should be noted that the example that is being used here is some sort of mechanical or physical product, but Reverse Engineering can be used for anything including software, electronics, houses, cars, and more. However, the techniques are different depending on the product.

Uses of Reverse Engineering

Before executing the Reverse Engineering process a well-planned life cycle analysis and cost/benefit analysis should be conducted to justify the reverse engineering effort. Reverse engineering is typically cost effective only if the products involved reflect a high investment or will be reproduced in large quantities. A common misconception regarding Reverse Engineering is that it is only used for stealing or copying products.

Examples of the different uses for Reverse Engineering are listed below.

- To understand how a product works.
- Investigate and adjust for errors and limitations in existing programs.
- Study the design principles of a product as part of an education in engineering.
- Making products that are compatible with existing systems.
- Evaluating a company's own products to understand limitations and advantages.
- Renewing and improving user interfaces.
- Translating a program from one language to another.
- Transforming obsolete products into modern versions for continued useful life (RODRGUES).

Additionally, some examples of problems overcome with the successful use of Reverse Engineering are listed below.

- Re-documenting of programs and relational databases.
- Identifying reusable assets and components.
- Recovering architectures of a program.
- Building traceability between code and documentation.

Although software reverse engineering originated in software maintenance, its definition is sufficiently broad so as to be applicable to many problem areas. One such example of this would be for testing purposes or to audit security and vulnerabilities. The practice of reverse engineering is widespread throughout the software industry. However, it also facilitates hackers to some extent.

Software companies fear, and rightly so, that their trade secret algorithms might be in danger because of this process. The methods are usually covert through external machines and they can expose a product in great detail. However, there is no law against Reverse Engineering. This is quite simply because Reverse Engineering has many legitimate uses within the engineering community.

The Reverse Engineering Process

The basic Reverse Engineering process has been divided into three steps that are listed below.

- Digitizing.
- Data Segmentation.
- Data Fitting.

The first objective of the Reverse Engineering methodology is to generate a conceptual model from a physical model. In this scenario, 3D-scanning and digitizing techniques aided by specialized software are used for model reconstruction. 3D-scanning and digitizing are processes for gathering data from an undefined three-dimensional surface. During the scanning process, an analogue scanning probe moves back and forth (contact or non-contact) across the article. During this process, the system records information about the surface in the form of numerical data by recording 3D-coordinates. This data may then be

used to create a CNC-program to machine a replica or geometric variant. Contrarily, the data can also be exported in various formats to a CAD/CAM system for further processing.

Factors for Digitizing Parts

- Model materials.
- Physical condition of the model.
- Need for fixtures.
- Alignment requirements.
- Digitizer errors.
- Available digitizers.

Difference between Digitizing and Scanning

The terms digitizing and scanning are often used to describe the same process. Traditionally, the term digitizing referred to the process of taking discrete points from a surface using a touch-trigger probe. However, with the introduction of new technologies in data capture such as laser, camera, vision, and analogue probe systems, the term digitizing is now used as the generic description for the process of acquiring data from undefined surfaces (M. Sokovic, 2005).

Reverse Engineering and Automotive Mechatronics

The term "reverse engineering" has its origin in mechanical engineering and describes the process of analysis of hardware by somebody else other than the engineer of a given product. However, Reverse Engineering was applied to enhance products or to analyze a competitor's products. Additionally, according to Cifuentes and Fitzgerald (2000), another term used, when associated with software, is "reengineering," which not only refers to the process of analyzing software alone, but also its translation into an understandable form.

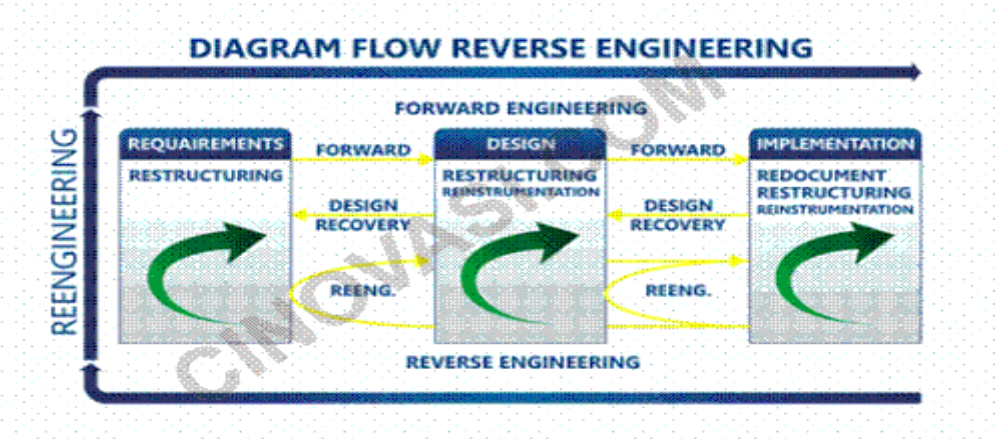


Figure 18.2 Reengineering Example

Furthermore, the two authors summarize the different types of software Reverse Engineering. They classify the types as black and white box Reverse Engineering. Black box Reverse Engineering usually looks at the behavior and functionality of a program and its documentation, if it's available, without internal examination. While white box Reverse Engineering involves looking internally to determine how the code performs.

Overall, reverse engineering of software focuses on the six areas listed below.

- Coping with the system complexity.
- Generation of alternative views.
- Recovery of lost information.
- Detection of side effects.
- Synthesis of higher abstractions.
- Facilitation of reuse.

These targets, that were originally defined for software Reverse Engineering, can also be transferred, to a certain extent, to the reverse engineering of automotive mechatronic systems and hence to the remanufacturing of those systems. However, for this to take place there are a few considerations. First, remanufacturers will have to cope with the complexity of mechatronic systems. "Cope," in this case, means in this context, that it must be possible to operate an automotive mechatronic system independently from its original environment (the vehicle).

Second, is the consideration of universal taxonomies. They have to be detected in order to transfer the gained knowledge to similar mechatronic systems or to other variants of the system. This is especially so with the high degree of variation of similarly looking mechatronic systems and control units that make it difficult for the remanufacturers to manage the complexity of automotive components that can usually differ by only slight margins.

Third, recovery of missing, rather than lost, information can be one of the most important aspects of the remanufacturing process. With these factors in mind, it must be understood that progress is not possible without its challenges, but it is achievable. The increasing complexity and variety of mechatronic devices cannot be handled with traditional methodologies. Therefore, remanufacturing companies have to build up new Reverse Engineering expertise, find methodological innovations, and develop new technologies that especially focus on the tasks, testing, and diagnostics of automotive systems and their subassemblies (Käufl).

Program Analysis and Applications

Years past have seen the development of several program analysis techniques and tools. Some of them rely on static analysis techniques, while recent years have seen an increasing use of dynamic analysis as an effective way to complement static analysis. However, it has been shown that dynamic engineering can be expensive and time consuming. With that said, it is necessary for dealing with many Reverse Engineering problems when static analysis does not suffice. Currently, several analysis toolkits are available to help reverse engineers in their tasks. For example, the Design Maintenance System (DMS) developed by Semantic Designs, the TXL, or the Stratego toolkit.

In a software environment, these toolkits offer options for parsing the source code and the ability to perform rule-based transformations. Despite the effort that has been put into the development of source code analysis tools, and despite the maturity and efficiency of parsing technology, the diffusion of a wide number of programming languages, a phenomenon known as the “500 language problem,” or problems such as dealing with macros and reprocessed directives, highlighted the dire need for alternative source code analysis approaches. Thus, Moonen proposed the idea of source code analysis through two processes he coined as island parsing and lake parsing that ultimately analyze only the code fragments that are relevant for a particular purpose.

Today's development environments such as Eclipse or NetBeans strongly favor such an integration concept. In fact, they permit the development of tools, including Reverse Engineering tools, as plugins integrated into the development environment capable of interacting with other tools, such as the source code editor, and capable of accessing the source code file a programmer is currently writing. Additionally, a substantial amount of work on program analysis was completed to deal with peculiarities introduced by object-oriented languages (e.g. polymorphism).

An intriguing development that has inspired a great deal of successful research is the presence of clones in software systems. The outcome was the production of different processes like token-based, AST-based, and metrics-based. Each one these has different advantages, such as high precision (AST-based) or high recall (token based), language independence, or the ability to detect plagiarism (metric-based).

Empirical studies have been carried out to analyze the presence and the evolution of clones in large software systems. Cloning percentages tend to remain stable because new clones tend to appear as fast as old ones disappear. Additionally, contrary to common wisdom, it has been found that the presence of clones is not necessarily harmful. Provided that the individuals responsible for maintenance are aware of their presence, clones constitute a widely adopted mechanism to facilitate software development. Clone removal, on the other hand, may be risky and undesired.

Aspect oriented programming is a programming approach that represents one of the new frontiers of software development. It addresses the issue of crosscutting concerns. Crosscutting here means features spread across many modules with a new modularization unit that is the aspect that encapsulates them. To support the maintenance of crosscutting concerns, as well as to refactor them into aspects, it is needed to identify them in the source code. With this in mind, several aspect mining approaches have been developed. They are based on the analysis of method fan-in or dynamic analysis of execution traces.

Yet another useful area of analysis is the text in a software product, either in the form of comments or identifiers. It has played a central role in Reverse Engineering. In particular, it is usually evaluated to recover traceability links between different software artifacts, using Vector Space Models and Probabilistic Ranking or Latent Semantic Indexing (LSI). Also, textual analysis using Information Retrieval techniques has been used to perform a software

quality assessment based on the similarity between identifiers and comments to measure the conceptual cohesion of classes or to perform semantic clustering.

Future Trends in Reverse Engineering

Any life cycle method in project management showing merit generally has a future and it is no different with Reverse Engineering. When a process stands the test of time, it can be counted on. The section below discusses future Reverse Engineering trends.

Future Trends in Program Analysis

One of the key challenges of program analysis for today and tomorrow is to deal with the dynamic nature of innovative integrated components. Many programming languages widely used today account for this aspect, which creates a powerful development instrument, but in the end, it only makes analysis more difficult. For example, languages like Java introduce the concept of reflection and the ability of loading classes at run-time. This affects many analysis techniques, like static points-to analysis.

With runtime class loading it is not possible to determine the set of objects a reference points to. This is often why dynamic analysis is required as an essential component of static analysis. On the other hand, methods like reflection can lessen the burden of analysis tasks. One example of this is providing access to fields and methods of a given class.

Another important program analysis challenge is cross-language applications. New program analysis tools were deigned to cope with advancements leading to diversity in languages and technologies used to develop a single software product. For example, some products are often composed of HTML fragments, server-side scripting, client side scripting, and database queries written in SQL.

New products will represent the source for future applications of Reverse Engineering. One example of this is represented by the need for analyzing artifacts produced by what Burnettet defined as ascend-user programming. Ultimately, this is just development of software by using productivity tools.

Reverse Engineering research has highlighted the duality between static and dynamic analysis and the need to find synergies between the two techniques. It is advantageous to exploit the advantages of both and limit their disadvantages. However, both static and dynamic analysis techniques can be used to analyze a software system configuration

individually. Regardless, it is obviously beneficial to understand how software versions change with each release. For example, do some releases change together, are release changes aligned with other software characteristics, and also are there other properties and characteristics of releases that should be detailed and investigated. This kind of analysis is feasible, all thanks to the wide use of versioning systems like Concurrent Versions System (CVS) or Subversion (SVN) and defect reporting systems such as Bugzilla.

With all of the intensive research that has been completed over the last few years, a new section of research emerged in the area of software repositories and the mining of them. Relevant work has been reported in software maintenance related conferences in the Mining Software Repositories (MSR) workshops and in major journals. While these kinds of studies were once completed as an effective way for studying software evolution based on developers' behaviors or to study the correlation among different software characteristics, recently software repository information has been used as an alternative or complementary way of software analysis. For example, Geigeret related clones with software repositories co-changes and Canforaet with crosscutting concerns.

During the last 10 years software analysis moved from a single dimension (static analysis) to two dimensions (static and dynamic analysis) and, finally, today's opportunity a third dimension consisting of the historical analysis of data extracted from software repositories. The data could include differences between file revisions and also relationships between bug reports and code repositories. Additionally, this data could further include the change rationale as documented in the fix reports and in CVS messages. All of this constitutes valuable sources of information complementary to static and dynamic analysis (A SRE METHODOLY FOR MODERNIZATION).

Advantages of Reverse Engineering

As with the rest of the life cycles, there are many advantages to Reverse Engineering that will be discussed in detail below. One obvious advantage pointed out above is that it is not solely a software project management life cycle. Further advantages are listed below.

- Useful if the original manufacturer is not producing the product any more.
- Useful if the design or documentation for the original product is not available.
- Useful if the original manufacturer no longer exists.

- Useful for replicating a product and improving some needed features.
- Useful for discovering advantages of a competitor's product.
- Useful for developing an improved product based on those already in the market place.
- Useful to establish competitive performance benchmarks in a market.
- Useful for redesigning an old product with new materials using new and improved manufacturing processes.

Disadvantages of Reverse Engineering

One of the main disadvantages of Reverse Engineering is that there are practical limits to the improvements that can be made. For instance, with software, it is not practical to write a functional approach and later decide to convert it to object oriented programming. Additionally, in a software realm, high costs can be incurred if architectural changes are made at a design level. Basically, as with every other decision in life it needs to make sense. There would be no reason to reverse engineer a part to modify it if the business case shows that it is better to end of life the product (Kalyankar).

- There are practical limits to improvements that can be made to existing products.
- A company could endure an expensive Reverse Engineering process and end up with nothing to show for it.
- Legal implications could restrict a company from using any information gained through the Reverse Engineering process to create another product.
- The business case may not support the Reverse Engineering operation.

Conclusion

Forthcoming Reverse Engineering techniques will rely on given information across three different and complementary dimensions. These are static analysis, dynamic analysis, and, as a new dimension, historical analysis of artifact evolution. Today and continuing in the future software will be highly dynamic, heterogeneous, and integrated.

In the future, Reverse Engineering will continue to be an essential part of development processes. Through its continued use over time, like other processes that survive, it will evolve for various purposes from consistency checking to design rationalization. Human interaction will, of course, continue to be fully integrated into the Reverse Engineering process and this suggests that the traditional dichotomy between automatic and semi-automatic Reverse Engineering will be overcome.

Traditionally, automatic Reverse Engineering has forced a trade-off between precision and recall, whereas semi-automatic Reverse Engineering has used human feedback to improve specific artifacts and the views produced, not the production process itself. It is possible for a new generation of Reverse Engineering techniques and tools with self-managing features to evolve.

The basic idea here is that engineer feedback is used to improve the Reverse Engineering process. As a system is reverse engineered, the engineer's feedback can be used to capture and store background and implicit knowledge on both the system and the engineers. Thus, the results of a system reverse engineering effort could improve over time not only in terms of precision and recall, but also in terms of the quality of the engineers work.

Building these features into future Reverse Engineering methods and tools poses many challenges to research. For example, how can background information and implicit knowledge be captured from the interaction with engineers? How could engineers' feedback be used to learn better ways of reverse engineering a system? What information needs to be captured in the interaction with an engineer to build a profile useful to guide future reverse engineering? How can captured knowledge of tasks be incorporated into a Reverse Engineering tool?

There is still yet another important aspect that is related to the role Reverse Engineering will play in the development process in the future. This is the diffusion of lightweight processes where requirement-to-code cycles represent a reality in today's software development practice. These processes however require the two key factors below.

- A continuous consistency check and alignment of software artifacts at different levels.
- Continuous refactoring.

The integration of Reverse Engineering tools into development environments can make this possible, enabling a tight form of continuous reverse engineering in which, during the forward phase, artifacts are continuously analyzed and, if necessary, transformed. Reverse Engineering will continuously present the developer with updated information, changing each time as the developer modifies the code and feedback is incorporated into the process. However, all of that said, on its own, the Reverse Engineering process generally improves the quality of the produced artifacts by exploiting developer feedback.

The adoption of Reverse Engineering, though, is still limited and can really only be increased through addressing two key areas. The first is better education at collegiate and professional levels. Second, while Reverse Engineering research is prominent and published research papers contain sound validation, further empirical studies comparing methodologies would be useful. The ultimate role of empirical studies is twofold. On one hand you have the assessment of existing Reverse Engineering theories and, on the other hand, you have the need for developing new theories that can be used to make sense of data and conclusions (NEW FRONTIERS OF REVERSE ENGINEERING, 2007).

All in all, Reverse Engineering is here to stay. It may have started out covert, but it has a future. That future is dependent on greater understanding by enlightened project management professionals.

Chapter 19

Structured System Analysis and Design Method

The Structured System Analysis and Design Method (SSADM) is a project management life cycle widely used for Government projects in the UK. It is a model that analyzes, transforms, stores, captures, and then disseminates product requirement information to successfully deliver products. In 1980, it was originally developed for a UK government office that is now called the Office of Government Commerce.

It was designed to give the software industry yet another framework for the delivery of software products. Its key purpose was to execute government projects by dividing them into several steps, procedures, and processes. Europe embraced it widely and it has been implemented by both public and private sectors.

Looking at the stages and steps of SSADM, people sometimes confuse it with the traditional Waterfall model as it too has a series of steps that must be performed in a particular sequence. It follows the life cycle framework of the model and covers all stages systematically. Project management and project feasibility becomes crystal clear when working with SSADM. It is in sheer contrast to the RAD methodology but it ensures quality and effectiveness of the system similar to RAD.

In SSADM each step is defined with a specific set of techniques and procedures to be performed. This includes rules and regulations for keeping track of all the stated requirements and processes for communicating the requirements. The information and procedures are in textual and graphical form to facilitate a better understanding of the step being performed at that stage.

Several CASE tools support and promote SSADM. It is a model very broad in scope that adheres to standards to assure quality and efficiency. It is known for implementing only features that are useful. This makes collaboration with other tools much easier through interactive and innovative interfaces. Additionally, functions, main menu, and command structures have been developed to further improve user experience and efficiencies.

Objective of SSADM

The Structured System Analysis and Design Method was ultimately designed for the advancement of project management and control and to promote a sense of training and learning among staff. Each and every step of the process can be computer based or assisted with the use of computer aided tools so that the cohesiveness of the project remains intact. This can be useful in such cases as human resources rotating through the project.

SSADM was developed with the objectives below.

- Ensure that projects can successfully continue without damaging effects should a loss of staff occur.
- Develop better overall quality systems.
- Improve the way that projects are controlled and managed.
- Allow more effective use of experienced and inexperienced staff and their development efforts.
- Make it possible for projects to be supported by computer based tools (e.g. computer-aided software engineering systems).
- Improve communication between participants of a project so an effective framework is in place (Structured Systems Analysis and Design Methodology).

History of SSADM

- 1980: CCTA assessed analysis and design methods.
- 1981: Consultants developed SSADM v1.
- 1982: LBMS later developed a LSDM, is their proprietary version.
- 1983: SSADM becomes mandatory for all information system development.
- 1984: SSADM Version 2 released.
- 1986: SSADM Version 3 adopted by NCC.
- 1988: Certificate of Proficiency launched, being accepted as an open standard.

- 1989: Euromethod approach, CASE products certification scheme launched in same year.
- 1990: Launch of version 4.
- 1993: SSADM V4 Standard and Tools Conformance Scheme.
- 1995: SSADM V4+ announced, V4.2 has been launched.
- 2000: CCTA renamed SSADM as "Business System Development."

1980's UK Government's Commissioning of SSADM

In the 1980's the Central Computer Technology Agency ((CCTA (UK)) became the first ever agency to initiate and complete an in-depth analysis to create a design process. At that time, Yourdon's techniques were widely in use and represented the standard for structured analysis of modeling. However, despite following the standard, there was a general perception of failure for various information technology projects. The large scale technology projects that several government agencies were in need of thus suffered from this same perception. Cost overflows were common and gaps between requirements and functionality revealed the need for a new standard and process.

The CCTA, which supervises various UK government projects, commissioned a proposal from the Learmonth and Burchett Management Systems group (LBMS) as a basis for the design of SSADM. LBMS actually beat out four other management firms and was able to create a solid design methodology based on their previous work. The final product was released in 1981 as the first version of SSADM. In 1983 SSADM was made compulsory for all government projects in the UK.

1990's – SSADM vs. DSDM and Other 4th Generation Tools

The 1990's saw rapid emergence of new software development methods that questioned the necessity of the "slow and thorough" approaches of the eighties. SSADM is considered one of these thorough methods of creating a product from the ground up. However, some of the newer development methods are known for bypassing several steps of life cycles like these for the sake of expediency. Agile and RAD (Rapid Application Development) make use of some of the facets of SSADM, but transition more quickly and

directly into the development phase. Keith Richards of KRS consultants cites the DSDM (Dynamic Systems Development Method) as being the alternative to SSADM techniques.

SSADM is considered another methodology born out of Waterfall while DSDM is viewed as a rapid development technique. Pat Phelan seems to agree with this and he argues that "SSADM (Structured Systems Analysis and Design Methodology) is based on the traditional Structured Programming techniques. It uses a formal design process that is a direct descendant of the "Waterfall" methodologies. This process tends to be relatively slow, but because the process tends to be exhaustive in both finding and debating every reasonable need, the results tend to be large and cumbersome, but thorough." Pat also acknowledges a major shortcoming of rapid application design in that the end product often does not perform well over time (Thomas Kwasa).

Criticisms of Classical System Analysis Approaches

Classical Methods Are Massive

Project documents are a large and often cumbersome required process for the first design phases. These specifications cover the entire product and therefore require that the developers understand them in their totality. This factor prohibited the emergence of design methods that sought a more modular approach to project management where team members may have sought an understanding of one of the building blocks of a product without necessarily needing to understand the others.

Classical Methods Contain Redundancies

Information is often repeated several times in different sections of project documents. Suppose, for example, if user requirements changed, modifications would need to possibly be made to several locations of a document or even several different documents. In the early 1960's, updates and revisions often proved to be more daunting in the absence of the word processing capabilities of today.

Classical Methods Usually Contain Inconsistencies

Redundancy and the need to revise several areas of project documents often created the related problems of documentation inconsistency. Several documents with common information that needs updating can often lead to documentation errors. One approach for combating this is the use of document mapping programs and document control departments.

Classical Methods Are Ambiguous

Different people often perceive information in different ways. Project management documentation is no exception. This leads to detailed requirements often being perceived differently among team members, which leads to the perception of ambiguity (Thomas Kwasa).

SSADM Techniques

Logical Data Modeling

Logical data modeling is where the division of data entities and their relationships take place. Each process for every entity and relationship is detailed separately. This allows the development of a product to have great flexibility. The data requirements of the product being designed are identified, modeled, and documented. This data is then separated into entities and relationships between these entities are identified.

Data Flow Modeling

This technique is more concerned with data flow with respect to the processes of information systems. It takes into account all the information about the data structure, data documentation, sending and receiving parties of the data, and the data flows specifying the flow path of the data. In short, the concern here is how the data moves through information systems and examining of processes, data stores, external entities, and data flows.

Entity Behavior Modeling

In brief, entity behavior modeling is the identifying, modeling, and documenting of events with respect to the entities in the system and the order that these events take place (Structured Systems Analysis and Design Methodology).

Overview of SSADM Structure

SSADM incorporates the usual phases of a project that facilitate the transformation from concept to prototype to finished product. The feasibility study is a high level summary of the project and its benefits. Additionally, it covers what the project hopes to accomplish in the future and the problems it is meant to solve. Cost benefit analysis is generally addressed at this stage of the process also.

Once the project has been approved, the requirements analysis phase begins. This is where an investigation of the technological and business options is undertaken. The purpose

of the requirements phase is to examine the objectives of the project and propose long term and cost effective solutions for delivering it. At this time, the team members acquaint themselves with the technical characteristics of the business goals while reviewing the data requirements and product's needs. Additionally, end users can be intimately involved in this phase of the development process. If this project is yet another stage of an existing product line, there will certainly be comparisons made between the functionality of the current product as opposed to the new product being proposed.

Product specifications are the focus of the next stage in the project management life cycle. At this stage, detailed graphical documents that explore and detail the relationships between systems are created. The logical system specification design starts and involves a proper cost benefit analysis of the hardware options. The logical design of the new system also begins to take place. Furthermore, the databases update and delete functions and the methods of handling system inquiries are all placed into the product model. Finally the physical design stage is completed where both the data and the processing options are converted into a design that will perform successfully.

SSADM Process Stages

SSADM is a project management life cycle and as such there are phases, or stages, in the life cycle process to follow. Those stages are detailed below.

Stage 0: Feasibility Study

Before approving any project, it is necessary to prepare a feasibility report for that project. Projects require the investment of extensive resources and a feasibility report details the benefits, risks, and opportunities of the project in real world terms. This report is a necessary tool for team and management review before deciding to execute the project.

That said, people still take shortcuts. This is to say that not all projects have effective feasibility reports prepared before execution. In projects where timelines are short and challenging, personnel don't pay much attention to the feasibility report and often subvert the process and jump to the next stage without giving their actions a second thought. Feasibility reports are typically documented formally and approved by executive management and/or clients.

Stage 1: Investigation of the Current Environment

It is said that this stage is the most crucial stage of SSADM. In this stage documentation, research, review of previous projects, and team input are all part of the project planning effort. Team members will typically start their detailed investigation for this stage with questionnaires gathering knowledge of past projects that are similar. Only by way of the process will the scope of the project be truly understood and required resources defined.

Stage 0 was all about is a given project even feasible. Stage 1 is about what would it really take in detail to complete the project and how much will it cost. Requirements are defined and hardened. Schedules and budgets are completed. This is also a good time for a SWOT (Strengths, Weaknesses, Opportunities, and Threats) analysis to see where the organization as a whole shines and where it may need help from outside resources. Risk planning is also an important part of this step as it is throughout the project.

Another component of the planning that takes place in this stage is Data Flow Diagrams (DFD). DFD's document logical systems, requirements, user catalogs, logical description of environments, and roles and responsibilities of developers and analysts. These are the deliverables of this stage. These DFD's are then compared to themselves to eliminate repetitive or redundant content for project efficiencies.

Stage 2: Business System Options

After conducting thorough research and assessing all possible outcomes and consequences of the project, the team evaluates several different past business options for executing the project. The identified options are investigated completely including any history the organization has with a proposed option. The team then brainstorms even more possible options. Either modify an old approach, implement a whole new approach, or a combination of the two.

Analysts will sometimes conduct brainstorming sessions among the potential users of the end product to understand their perceptions of the project. Collaboration of this type often proves useful in providing options for delivery of the project and the final product design. Results of sessions like these can effect such areas as the boundaries between users and the end product, cost benefit of the product, level of impact of the new product when implemented, level of usage and acceptance of the product within an organization, the kind of

users that would engage with the final product, and the level of performance required of the product. The ultimate deliverable for this stage is an agreed upon business option.

Stage 3: Requirement Specification

The requirement stage is considered by some to be the most complicated and delicate stage of SSADM. This is the stage where all the outputs of the prior stages are integrated and executed. Here, results of the very first stages are combined with the business option selected in Stage 2 and the product begins to take on a more definitive look and feel. At this point the product should be free from any ambiguities, inconsistencies, undefined features, or vague operations. In short, the product should clearly detail how it can and will deliver the expected end results.

As stated before, product specifications are created with the aid of DFD's. These data models consist of logical data structures that explain entity relationships. The diagrams, models, data descriptions, and their relationships with each other are essential to understanding the scope of the product.

Additionally, Entity Life Histories (ELH) are the description of all the significant events of the entity's history and Effect Correspondence Diagrams (ECD) illustrate the relationships and dependence of each event on each and every entity. These all help the end users to intimately understand the entity relationships and events that can occur and their effects on the relationships. This all leads to a better understanding of the requirements and additional requirements can be added later.

By the end of this stage the requirements catalog will have been updated. Furthermore, the function matrix, definitions of functions and processes, ELH's, ECD's, and logical data models have been updated. The final deliverable for this stage is one all-inclusive document package.

Stage 4: Technical System Options

Like a Business System Option, this stage's purpose is to select a technical option for successful project delivery. As with Stage 2, in this stage several different implementation options are formulated and proposed with the team agreeing on the best approach. Here again, the final approach could be any of the ones proposed or a combination thereof.

As stated, unlike with the Business System Options stage, the focus here is more technical in nature so the team must evaluate different areas. These include hardware and software, total cost of implementation, personnel skillsets required, time and space allocation, and the format of any possible interface required of the product. All variables must be in accordance with budget, time, and standards allocated and defined by the organization at the beginning of the project.

Stage 5: Logical Design

In previous stages of the project, the product structures were proposed and chosen while addressing their pros and cons of implementation and output results. This stage is all about logical design, which details the human interface of a product. This includes the command structure, menu layouts, and methods of product navigation.

During this stage, time has been dedicated to user dialogues to clearly and properly define them. These dialogues are the gateway of interaction between the user and the product. Furthermore, there are several other activities being performed during this stage. They are the analysis of data descriptions, consequences of events, descriptions of functions, effect of relationships, Effect Correspondence Diagrams, and event histories. So, the deliverable of this stage is comprised of logical data structures, data catalogs, a logical process model, and dialogs for users to interact with the product.

Stage 6: Physical Design

Stage 6, while named Physical Design, is actually the execution stage of this project management life cycle. In this stage all requirements, specifications, models, and processes are actually transformed into a finished product. It is the most technical and practical stage of SSADM.

Furthermore, at this stage all the logical models are being translated into real database structures and interfaces. Functions are being implemented by the hardware and software according to the size and space available for it. This operation is carefully conducted so that size will be optimized and pose no problem.

However, performance must not be compromised while optimizing the functions. This final physical design will be the example of what the proposed product must look like and how it must execute the functions it is required to perform. This design contains all minor and major details of the product.

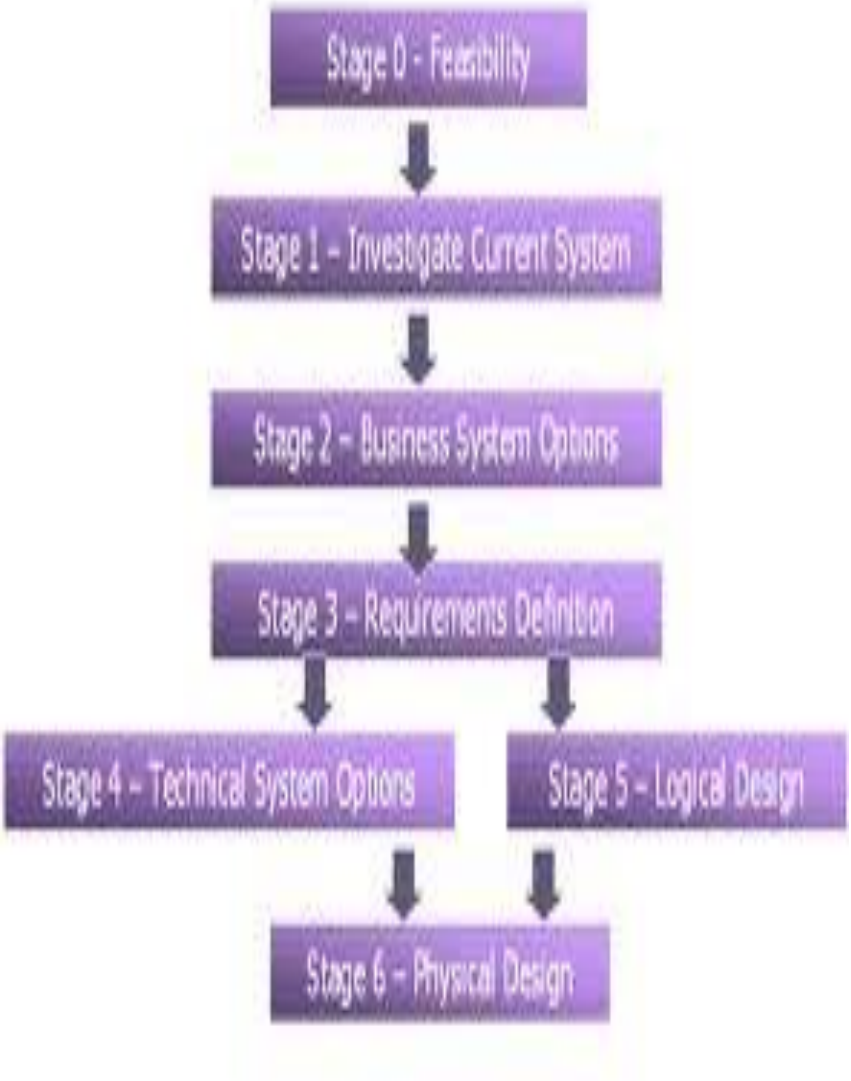


Figure 19.1 SSADM Life Cycle

Advantages of SSADM

SSADM has a modular structure that relates directly to the project deliverables and drives all aspects of project management. It includes clear specifications of what is to be produced and how it is to be managed and reviewed. Additionally, there are many well-defined management techniques employed with this project management life cycle. The following is a list of advantages of SSADM. The approach taken by SSADM to facilitate these benefits is also detailed below.

1. The process cleanly delivers products that meet user's needs by continuously involving the end users, prototyping business activities, and enhancing communication through diagrammatic approaches. SSADM improves the prospects of success of both large and small projects.

2. SSADM enables the delivery of products in a constantly changing, volatile business environment. Business Activity Modeling and Work Practice Modeling ensure that the focus of the project remains aligned with corporate strategy and requirements regardless of external activities. Additionally, project documentation spotlights the areas below for greater insights of the product's purpose.

    - Business objectives.
    - Users' understanding of the business objectives.
    - The relational link among the needs of the business and the product under development.
    - Detailed processes for the design, construction, maintenance, and enhancement of the product.

3. SSADM has been known to improve cost effectiveness by using some of the most common techniques leveraged by the worldwide market place. This includes the use of Data Flow Modeling and Logical Data Modeling. It promotes their effective use by aiding forward planning and enhancing the skillset base in the organization.

4. Quality is also normally improved with SSADM. Obviously, increased quality is accomplished by reducing error rates. This is achieved with the involvement of the complete team and skilled practitioners focusing intently on errors starting during the very early stages of the project management life cycle. Rigorous techniques combined with adequate checks for completeness and consistency promote accuracy. By defining the required quality of design documents and designing tests for them, SSADM promotes effective quality management.

5. SSADM is flexible and scalable. The ability to tailor a project management life cycle to different environments is key to organizational adoption and SSADM

delivers. This easily allows the reuse of human resources and skillsets on other projects.

6. SSADM delivers improved productivity. Major boosts in performance are achieved through the means listed below.
    - Documented techniques that accurately specify business and system requirements.
    - Defining what is needed in automated support tools to support SSADM techniques.
    - Using a product oriented approach and avoiding the need to perform unnecessary tasks or to produce documentation with unnecessary levels of details.
    - Encouraging quality criteria and specifications.
7. SSADM allows an organization to avoid being locked into a single product development approach with the separation of logical system specifications and physical design.
8. To avoid the bureaucracy often found in the development community, SSADM has been designed to provide useful tools for project managers to transfer expertise to experienced practitioners. The use of these tools has benefits, as well as costs efficiencies, that are visible to both management and end users (AN INTRODUCTION TO SSADM, Jan 2012).

Summary of Advantages of SSADM

- SSADM improves the prospects of success of both large and small projects.
- SSADM enables the delivery of products in a constantly changing, volatile business environment.
- SSADM has been known to improve cost effectiveness by using some of the most common techniques leveraged by the worldwide market place.
- Quality is improved through the use of SSADM.

- SSADM is flexible and scalable.
- SSADM delivers improved productivity.
- SSADM allows for multiple product development approaches.
- SSADM avoids the bureaucracy often found in the development community.

Disadvantages of SSADM

As detailed throughout this chapter, SSADM is a project management life cycle that does require a great deal of documentation. Thus each and every single aspect of the project is evaluated by the team with regards to risks, opportunities, scope, time, and resources. Compared to other life cycles, the requirement for thorough documentation and deep analysis leads to increased costs, extended timelines, and irritated team members as they focus on compliance with the standards and procedures. The extensive nature of the life cycle requires intimate knowledge of its operation for successful deployment. This generally entails training team members unfamiliar with the process and requires additional ramp up time.

Summary of Disadvantages of SSADM

- Extensive documentation required.
- Training team members on SSADM processes can increase project ramp up times.

Chapter 20

PRAMIS

Today's worldwide economic conditions are brutal. As such, companies are seeking out new ways to efficiently use their scarce resources to complete projects successfully and within their budgetary constraints. In order to achieve this objective, the PRAMIS project management life cycle is sometimes used. The PRAMIS methodology is most commonly known as the "Project Management and Management Accounting Methodology." It dictates the usage of management accounting principles by project managers to provide them with all the basic information to make efficient and effective business decisions. Additionally, it also allows project managers to conduct their planning, management, and controlling operations within the specified budget and limited resources of the project.

History of PRAMIS

During the industrial revolution, two industries specifically played a major role in the advent of management accounting principles to achieve strategic objectives. They were the textile industry and the railroad industry (Caplan, 2006). Textile mills needed to develop methods to track and control the efficient usage of raw materials and labor used in the manufacturing of fabrics and clothing items. The railroads needed such methodologies that could track and calculate the substantial capital investments required for the construction of roadbeds and tracks. Once the operations were underway, the railroad managers needed to handle large sums of cash receipts from various customers. Hence, they developed managerial and financial measures to obtain efficiency in moving freight and travelers.

Near the end of the 19th century, many new industries and businesses emerged in the developed nations like the United States of America, Great Britain, and many others. These new industries included steel manufacturing, consumer products, manufacturers of food items, and large retail institutions like Sears. All these industries developed new and improved accounting systems in order to control and monitor their business functions.

Later, in the 20th century, management accounting principles emerged right alongside the field of industrial engineering. Industrial engineers were in dire need of such methodologies that could control production of specialized items that needed specific standards for measuring inputs of materials and labor within the time constraints and compare

various results with each other. This caused the emergence of standardized costing systems, which are still used by manufacturing companies to handle projects. These systems then evolved into bigger and better management accounting concepts and techniques throughout the 20th century. Many of them are still well established today.

Business historians suggest that the companies that were "innovators" in adopting management accounting methods were DuPont, General Motors, and General Electric. However, it should be noted that management accounting did not become a part of regulatory operations until the implementation of the Foreign Corrupt Practices Act of 1977, which stated that large organizations must maintain ample systems to manage internal control. Nowadays, companies invest an immense amount of time and energy in establishing adequate management accounting systems for their organizations to achieve their unique objectives.

What is PRAMIS?

PRAMIS, as discussed before, is basically management accounting. Its focus is to provide project managers with accounting information that allows them to better understand and manage planning, scheduling, and controlling of project activities in an effective and efficient manner. To better understand this concept, below is the concise definition of management accounting by The Chartered Institute of Management Accountants, United Kingdom (Accountants, n.d.).

"Management Accounting is an integral part of management concerned with identifying, presenting and interpreting information used for formulating strategy, planning and controlling activities, decision-making, optimizing the use of resources, disclosure to shareholders and others external to the entity, disclosure to employees, and safeguarding assets."

It should be taken into consideration that management accounting falls under the scope of job order costing (Vitez, 2014), which is a system used for assigning manufacturing costs to products or batches of products that are sufficiently different from each other (Averkamp, n.d.). With job order costing all the costs associated with the unique individual projects can be tracked. As discussed previously, companies make use of projects to track the progress and success of individual activities. Therefore project managers always need information on the financial performance of these projects, which can be provided with the PRAMIS methodology.

Construction, manufacturing, and industrial engineering firms are more efficient when adopting PRAMIS. For example, an airline manufacturing company receives special orders for unique and exclusive airplanes, so the company will need to create distinct accounting systems for each and every order. One particular system cannot be applied to all. If the company fails in managing the individual project costs, it will most likely result in cost overruns and lower profits.

The downside to using PRAMIS is its susceptibility to fraudulent activities (Vitez, 2014). Companies have been known to shift expenditures of a costly project to another in order to show a profit on a specific project. Of course, this falls under the category of fraud, as projects can have costs associated with them that do not give the true picture of the labor and materials used on them. In this scenario, the role of management is very crucial for approving labor and material costs on specific projects, which ensures that project managers don't misallocate costs or hide them to misrepresent the performance of projects to executive management.

Comparison of PRAMIS with Financial Accounting

In order to thoroughly understand the concept of PRAMIS, below is a comparison with financial accounting. Both of them are closely interrelated because PRAMIS or management accounting is rooted in financial accounting and both deal with monetary data and statistics. However, regardless of this close interrelationship, there are vast differences between the two and study of their differences is necessary to understand the concepts clearly.

Financial Accounting	PRAMIS
It follows the GAAP (Generally Accepted Accounting Principles) within America and other standards in other countries.	It generally follows no standardized principles. Companies usually develop management accounting standards according to their own specific needs for their unique projects.
It mostly focuses on calculating and reporting past performance of the company.	It mostly focuses on calculating, estimating, and reporting future transactions and activities of the company.
It emphasizes the reliability of the information	The data provided can be estimated

provided through objective measures.	through many subjective measures.
The information produced is used by external parties such as the shareholders, creditors, and the general public.	The information produced is used within the organization such as the functional departments, project managers, and employees.
A specific format must be followed for maintaining financial records so as to allow comparisons with other companies' financial records.	No specific format is needed to be followed for maintaining managerial accounting records as they can be of both formal and informal nature.
It provides quantitative information and monetary data.	It provides both quantitative and qualitative information along with monetary and non-monetary data
Financial accounting reports are usually presented annually, semi-annually, or on a quarterly basis.	Reports are presented as required by management whether daily, weekly, or monthly.
Maintaining these records is necessary.	It is not necessary to maintain management accounting records.
It deals with the organization as a whole, rather than handling individual components of the business.	It focuses on specific and distinct areas/units of the business, like the performance data on particular products, separate shops, departments, or divisions of the company.
It follows the accounting process of recording, organizing, and summarizing the overall financial information of the organization for analysis and interpretation purposes.	The required data for management accounting is retrieved from financial statements and costs are analyzed and interpreted.

Table 20.1 Financial Accounting Compared to PRAMIS

Characteristics of PRAMIS

PRAMIS is not constrained to one particular discipline of either management or accounting. It covers a vast array of tools and techniques for analysis of management decisions that improve the efficiency and attainment of organizational goals. The characteristics or features of PRAMIS are discussed below.

- **Attainment of Project Objectives:** The laws of project management are quite simple and are known as the triple constraints of project management. They are

time, scope, and resources. PRAMIS provides project managers with such information that assists them in achieving these key objectives. Historical data is used for formulating plans and then actual performance is compared with that data to give an accurate picture of where the organization and the project team stands in delivering those objectives successfully.

- **Increased Project Efficiency:** Efficiency is achieved by developing clear cut goals for each stage of the project. The performance statistics provided by PRAMIS can enable the project team and managers to target all identified inefficiencies and follow-up with plans for corrective action.

- **Better Decision Making:** PRAMIS, in its truest form, simply provides the project management team with all the essential information needed to base its decisions on and make the process easier for them. Historical data of similar projects and comparisons with current performance can present the management team with vital decision making information.

- **Provision of Accounting Information:** PRAMIS basically depends on accounting information, which is provided by the financial accountants of the organization. This information is then used for analytical purposes and for creating the basic policy decisions that the project depends upon. It presents this accounting information in such a way that fulfills all managerial needs on all levels of the project.

- **A Two-Way Relationship:** PRAMIS not only presents us with facts and figures but also investigates the reasons for the results. The results are then analyzed and interpreted and actions are taken for continuous process improvement.

- **No Hard and Fast Rules:** Unlike financial accounting, there are no standardized rules and regulations for developing PRAMIS tools for a project. PRAMIS tools can vary from one organization to another and they can even be different between various projects within the same company. They are developed on a subjective basis, therefore they are basically dependent on the people who use them and are also reliant on the organization’s policies and regulations for developing and applying such tools.

- **Usage of Distinctive Tools and Techniques:** The accounting data is more useful through the usage of special tools and techniques in PRAMIS. These techniques are determined according to the nature and status of the project. The tools and techniques include various costing techniques, statistical analysis, cash flow analysis, budgetary control, and many others that will be discussed later in this chapter.

- **Provides Information, Not Final Decisions:** Making final decisions is the job of the project manager and executive management of the company. It is not a part of PRAMIS processes. PRAMIS is a methodology that merely provides data for analyses and interpretation for making better decisions regarding the project.

- **Provides Forecasted Data:** PRAMIS can give management a view into the future of a project. Past data is analyzed through different tools and then forecasted data is provided regarding the particular effort. Forecasts can be created for production, sales, profits, and costs of the project.

Scope of PRAMIS

PRAMIS and/or management accounting deals with the presentation of accounting data (Malik N. S., 2013). It embodies in itself many disciplines of management, accounting, and business operations. Quite simply, the scope of PRAMIS encompasses the areas below.

- **Financial Accounting:** PRAMIS processes do not create basic financial data. Instead, it relies on financial accounting to provide historical data that acts as the basis for analysis and interpretation of accounting information necessary for all planning, controlling, and monitoring purposes within the project. Hence, it can be rightfully said that PRAMIS is dependent on financial accounting.

- **Cost Accounting:** Cost accounting provides PRAMIS with basic cost related data. It uses various methods for ascertaining expenses. Project managers then analyze the cost data and extrapolate necessary information for controlling and decision making with regards to the project. Budgetary control, cost analysis, inventory control, and all related techniques are used in conjunction with PRAMIS.

- **Tax Accounting and Tax Planning:** Analysis of all effects and implications of tax provisions on all projects, current and future, fall under the scope of PRAMIS. Taxable income and tax obligations are also determined. Therefore the person in charge of executing PRAMIS methodologies must have knowledge of tax laws and tax planning so that they can help to minimize the tax burden of the organization.

- **Inventory Control:** During the execution of projects, inventory must be controlled. For this purpose the management team should determine varying levels of stock available for the project from time to time.

- **Budgeting and Projecting Future Estimates:** PRAMIS makes use of tools that provide forecasting data and budgeting information regarding the project. With the help of forecasting and budgeting, management can determine the budget required, what budgetary controls must be exercised to keep the project within limits, and what policies and goals must be implemented.

- **Statistical Techniques:** Graphs, charts, samples, diagrams, and various other instruments are used with PRAMIS, which make information more comprehensive and understandable for planning and controlling purposes.

- **Management Information Systems:** Management Information Systems are highly sophisticated computer networks that provide extremely accurate real time data about the project and its progress.

- **Internal Audit and Control:** PRAMIS is highly dependent on the internal audit and control systems of an organization that are used to assess the progress and performance of a project.

- **Office Management Systems:** Emails, faxes, files, internal networks, and all such related office management systems also fall under the umbrella of PRAMIS.

- **Legal Provisions:** Knowledge of local laws and regulations of the country and/or area of operations of the business is a must for the organization so that management can ensure activities are in line with said boundaries.

- **Miscellaneous Areas:** In addition to the specific areas detailed above, the managerial areas of human resources, social impacts, environmental and inflation analysis, and all other related activities are covered under the scope of PRAMIS. This shows the immense impact the discipline has on every vital aspect of an organization.

PRAMIS Tools and Techniques

PRAMIS, or management accounting techniques, provide business leaders with tools that allow them to measure, analyze, and increase the profit margins of their projects while decreasing operating expenses and simplifying managerial tasks (DeBeneditti, n.d.). The various tools and techniques used with PRAMIS are listed below.

- Financial Statement Analysis.
- Working Capital Management.
- Budgetary Control.
- Responsibility Accounting.
- Throughput Accounting.
- Management Information Systems.

Each is discussed in greater detail below.

Financial Statement Analysis

A successful project can only be executed by a financially sound company. Therefore, the main tool of PRAMIS is a solid financial statement analysis. The statements below are included in the review and evaluation that reveal a basic understanding of a company's financial standing.

- Balance Sheet.
- Profit and Loss Statement.
- Cash Flow Statement.
- Funds Flow Statement.

The main focus of financial statement analysis is to validate the reported financial position of an organization. This includes performing an analysis, making adjustments for discrepancies, and finally conducting a financial ratio analysis on the adjusted and re-evaluated statements. The common tools for performing these reviews include comparative financial statements, trend analysis, ratio analysis, and common-size statements. All of these provide information about the profitability, solvency, liquidity, and stability of the company.

Profitability entails a review of the profit margin of an organization as well as its ability to sustain those profits in both the long and short term. This information is basically based on the Profit and Loss Statement. This statement summarizes revenues and expenses incurred by the company during a specific time period.

When the solvency of a company is interpreted and analyzed, it is revealed just how much the company is able to pay off its liabilities to its creditors and other third party institutions. These liabilities may include bank loans or other related instruments. Liquidity, on the other hand, is analyzed by reviewing a company's ability to convert its assets to cash as quickly as possible. Both of these are based on the data provided from the balance sheet of a company. In short, when the stability of a company is analyzed, it tells us the company's ability to continue operations for the long term without incurring substantial losses during the course of business. This assessment depends on the balance sheet, the profit and loss statement, and other financial and non-financial documents, reports, and statements.

Working Capital Management

In order to effectively execute a project, a company must be able to efficiently manage its current assets and liabilities. Therefore, proper management of working capital ensures the company will preserve adequate cash flow to meet its short term accountabilities and operating expenses. In dealing with working capital management, a working capital cycle is constructed that designates the amount of time required by the company to convert its current assets and liabilities into cash. If the company wants to increase the effectiveness of its working capital, then it must keep this cycle as short as possible. The four basic components of the working capital cycle are listed below.

- Cash.
- Creditors.

- Inventory.
- Debtors.

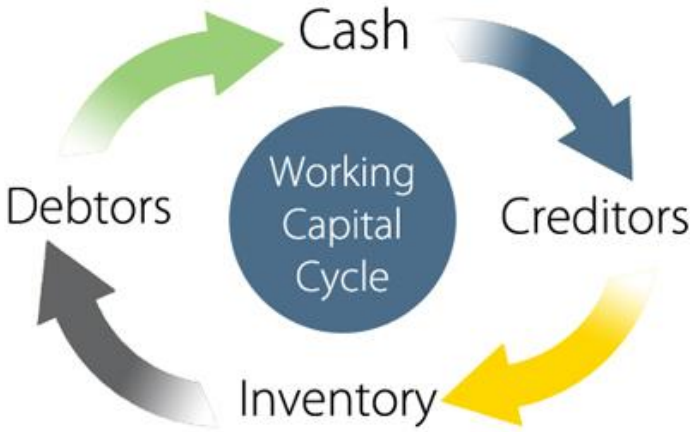


Figure 20.1 Working Capital Cycle

Strategies that help with shortening the working capital cycle include reducing the raw material inventory held, delaying payments to suppliers, reducing the amount of credit given to customers, and reducing work in process (WIP) by improving development techniques. Effective working capital management is achieved by using a few key performance ratios. These ratios can assist the management of a company with identifying highs and lows in the areas of accounts payable, accounts receivable, cash management, and inventory management. The ratios are listed below.

- Current Ratio: = Current Assets / Current Liabilities
- Quick Ratio: = Current Assets – Inventory / Current Liabilities
- Working Capital: = Current Assets – Current Liabilities
- Inventory Turnover Ratio: = Cost of Goods Sold / Average Inventory where, Average Inventory = (Beginning Inventory – Ending Inventory) / 2
- Debtor Days: = (Average Accounts Receivable / Sales) x 365
- Creditor Days: = (Average Accounts Payable / Cost of Sales) x 365

Budgetary Control

Budgetary control is straight forward. Quite simply, the actual budget of a project is compared to the planned or estimated budget. This allows management to determine if the project budget is within control limits. Budgets are created with the help of historical data and forecasted expectations. This technique involves the framing of the project budget, determining any deviations from the actual and estimated budget by computing variances, and also the implementation of corrective measures.

Responsibility Accounting

Responsibility accounting can be defined as below.

"It is a system of control where a responsibility is assigned to different executives of a concern for control of cost or increase of revenue. It is one of the basic components of a good control system. In this system, an executive is held responsible only for those activities for which he/she has been delegated a responsibility" (Bhattacharyya, 2010).

The basic underlying meaning of the above definition is that organizations can be complex and diversified entities and it is not possible to manage them as one single unit. Hence it is important to decentralize an organization into separate manageable parts. These separate units are referred to as "Responsibility Centers." These responsibility centers include the areas listed below.

- Revenue Centers.
- Cost Centers.
- Profit Centers.
- Investment Centers.

During the course of the project, responsibility is assigned to the areas with the greatest amount of impact on the effort. With the help of responsibility accounting, management can prepare annual, bi-annual, or monthly budgets for each responsibility center. In due time, the actual transactions are recorded by each responsibility center and a monthly report is prepared for final analysis. These reports represent the variances between the actual and estimated budgets allocated to each center. This tool is extremely useful in monitoring

the performance of the designated manager of each responsibility center and drives performance feedback.

Throughput Accounting

Based on Goldratt's Theory of Constraints, throughput accounting is a newly developed decision making tool that is an important part of PRAMIS. This tool identifies weak aspects that constrain an organization from reaching its desired goals. It then, consequently, focuses on those techniques that drive the behavior towards those key areas that help in successfully achieving the organizational goals.

It changes the way an organization thinks internally about its revenue generation, expenses, and profitability. Therefore, it changes the figures to be used for final decision making (Goldratt, n.d.). By leveraging throughput accounting, a company can focus its efforts towards those projects that truly generate more money, make better judgments about what investments are most profitable for the company, and get clearer comprehension about all the sub-systems of the project. This provides management with a better understanding of the system as a whole and also provides a more realistic reporting of the efficiency of the system in relation to its ultimate goal. According to the Theory of Constraints, three measures are used for decision making, which are also essential in throughput accounting. They are applicable in the exact order listed below.

- Throughput (T) – The money generated from the sale.
- Investment (I) – The money devoted to the effort.
- Operating Expense (OE) – The money used to turn the investment into sales.

All decisions can be made by basing them on the evaluation provided by the effect of the desired project on Throughput, Investment, and Operating Expense.

Management Information Systems

As stated above, Management Information Systems, or simply MIS, are computer based networks that provide managers with the tools to organize, assess, and manage all departments of an organization. MIS provides support systems, project management applications, databases, and hardware resources that help in decision making. This is because they provide past, present, and future predictions of information acquired by the company and also enable every department to run efficiently and effectively.

MIS enables companies to highlight their strengths and weaknesses by the accurate and timely analysis of revenue reports and performance records. It also provides readily available client and customer data and feedback that allows a company to better serve their customer's unique needs. Lastly, MIS can provide a competitive advantage in the marketplace.

Advantages of PRAMIS

PRAMIS has the advantages listed below.

- PRAMIS makes future decisions based on historical data. Therefore it is forward looking and progressive in nature.
- Increases the efficiency of each and every operation in a company because of its evaluation and comparison techniques based on pure performance.
- PRAMIS techniques can reduce a company's operating and capital expenditures, subsequently generating higher profits.
- PRAMIS provides simpler interpretation of financial reports. This simpler and comprehensive understanding allows management to make better decisions for the company's progress.
- PRAMIS processes allow for easier control of corporate cash flows. This is due to comprehensive studies of where revenue is derived from and where it is invested.
- PRAMIS is flexible. It can accommodate cycles on a monthly basis, a weekly basis, or whenever desired.

Disadvantages of PRAMIS

PRAMIS has the disadvantages listed below.

- It is highly dependent on accounting data. Therefore the accuracy of PRAMIS' results depends on the accuracy of the accounting data.
- Since it follows a subjective approach, top management can sometimes makes biased decisions and "tweak" the results in ways that benefit them rather than other stakeholders.

- PRAMIS is still in an evolutionary stage and is still maturing. The tools and techniques used give varying and fluctuating results.
- A vast knowledge of a number of subjects is required in order to completely understand PRAMIS. Without sufficient knowledge in subjects like accounting, statistics, economics, and management, then the ability to fully utilize this project management life cycle can be limited.

Chapter 21

Open Source Software Development

Open Source Software Development (OSSD) has changed the nature of software project management immensely. The most important aspect that makes it different from the traditional methods is the fact that the source code for the software is available to the general public. They can examine, evaluate, and alter the source code according to their requirements. It is certainly one of the most revolutionary processes in software development to date. Additionally, it has also opened doors for individual developers to develop their own software and work more efficiently. However, another aspect of OSSD is that there is no 800 phone number to call when you have problems with it. For software to be open source it must have the following characteristics, which will be discussed in detail later in this chapter.

- Free Redistribution.
- Source Code Available Publicly.
- Derived Works Created.
- Author's Source Code Integrity Maintained.
- No Discrimination Against Persons or Groups.
- No Discrimination Against Field of Endeavors.
- Distribution of License Rights.
- License Must Not Be Specific to a Product.
- License Must Not Restrict Other Software.
- License Must Be Technology-Neutral.

Introduction to Open Source Software Development

Open Source Software Development gained popularity in 1997 when software developers started modifying their informal developmental model and provided source code along with their software. This method helped researches, writers, developers, and educators figure out a way to work collaboratively. Ultimately, OSSD can be described as a systematic

process to harness open development of code that encourages peer review and low cost design. The code, once released, is then managed by a community instead of one person or a company.

Open source software is usually created by one developer, or a group of developers, who write code that is later made available for peer review and refinement. The source code goes through mass peer reviews, which makes debugging and rectifying errors much more efficient. That said, Open Source Software Development is an incremental process. The increments are completed rapidly since the review process is rapid and effective (Riley, n.d.). One reason the Open Source Software Development process became successful was because of the wide use of the internet making communication fast and easy.

Open source did not just become popular because of rapid peer review and modifications though. One of the main reasons for its growth was the GNU public License (GPL). GPL can be described as the terms and conditions of the open source software that are attached to it and are followed in all situations. The license allows any individual to change, modify, evaluate, and redistribute the software given that two basic rules are followed; a) the software should have the GPL license and author's notes in the original source code and b) the receivers of the source code should be allowed redistribution rights (Riley, n.d.).

With all the advantages of OSSD though, there are still perception issues that hamper its acceptance. Where the business world sees it as an easy and cost saving approach to software delivery, most commercial vendors see it as being an unreliable source of development that lacks the ability to produce desired results. Also, open source developers are sometimes perceived to lack competency to develop high quality software compared to their commercial competitors (Prehn, 2007).

Irrespective of all the criticism, the use of this open source development methodology has resulted in the creation of many successful software applications and operating systems. As such, it is now recognized as an alternative to traditional software design methodologies. Even companies like Netscape have reported to have increased market share after applying the open source development methodology. As a result, many new developers tend to opt for open source development (Prehn, 2007).

Furthermore, many companies developing software opt for open source development thereby supporting the idea that the software created does not always lack quality. Some of

the more well-known names in this realm are JUnit, Eclipse IDE, Apache Webserver, JBoss Application Server, Apache Tomcat Container, and several Linux distributions (Prehn, 2007). Some estimates have even placed consumer savings at nearly $60 billion per year due to the use of OSSD.

As good as OSSD is though, not all software can be developed using this process. As with every other process in this book, it depends entirely on the type of software being developed to decide on the best methodology. Therefore, one key factor for opting for open source development is that the project has to be unique in nature and benefit from peer review (Prehn, 2007).

Interestingly, when it comes to actually defining the open source process no one has been able to propose a definition 100% agreeable to the community. Many people have their own thoughts on how it should be defined. For example, Eric Steven Raymond has described the comparison between the classic software development methodology and open source methodology as "The Cathedral and the Bazaar." The classic model is the Cathedral as it is thoroughly planned with predefined hierarchies and responsibilities. Alternatively, open source is termed as the Bazaar where everyone has their own ideas, agenda, and strategies. A Bazaar, of course, being a place where everything is happening at once and thus creating chaos (Prehn, 2007).

Essentially, the differences between the traditional and open source models are listed below (Prehn, 2007).

- Open source projects are built by hundreds or even thousands of potential developers.
- Developers choose to work based on their own free will. Work is not assigned by anyone.
- There are no predefined rules and regulations that need to be followed.
- There are no milestones or lists of deliverables.

History of Open Source Software Development

The concept of open source development is rumored to have originated in the early 1960s. This was perhaps the earliest attempt by developers to share code and get it evaluated

by fellow developers. According to Eric Raymond and Tim O'Reilly, open source software was originally just named "Free Software" in February 1998 (Aurum, Ghapanchi, & Aybuke).

The best way to understand Open Source Software Development is to compare it with Closed Source Software development (CSS). In CSS the source code for the software is a precious secret that is protected by laws, but in OSS (Open Source Software) the source code is readily available to the general public. When developing with CSS a company employs professional developers to create their product. However, with OSS, developers volunteer for the project and the code is available for free. The eventual outcome of OSS is a product that lets everyone view and use the code to their benefit. As a result, generally speaking, OSS can more easily meet a consumer's needs due to flexibility (Aurum, Ghapanchi, & Aybuke).

However, while access to source code in the open source software environment is generally available to everyone, in some cases, access is still restricted to a limited group of people. One good example of this case is Microsoft even though they are not generally considered open source. Their source code is available, but only to a few selected people, whereas, Linux provides their code to everyone without limiting the access. Furthermore, any code released with the open source development methodology must be approved by OSI (Open Source Initiative). (Aurum, Ghapanchi, & Aybuke).

A large number of research efforts have been conducted on the open source development methodology to vet it and it has been generally concluded that this process has the potential to grow further. It is widely used by millions because, as stated above, the development process is independent and the developer can work according to his/her own schedule and desire (Aurum, Ghapanchi, & Aybuke). Mozilla, Apache, UNIX, Linux, and even Perl are examples of successful open source development (Aurum, Ghapanchi, & Aybuke).

Essentials of Open Source Software Development

Open source development believes in a process that is distributed amongst numerous developers working in multiple locations. These projects are successful when the team is informed and aware of all the technicalities involved, everything is well-documented, and small, frequent increments are released to identify potential bugs earlier in the project life cycle (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011). In fact, one of the key

benefits of the Open Source Software Development model is that only one individual or a set of individuals are responsible for the development and maintenance of the code. The code elements contributed by one or more developers into the main code repository is then evaluated to ensure that it meets the overall standards and vision of the project (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

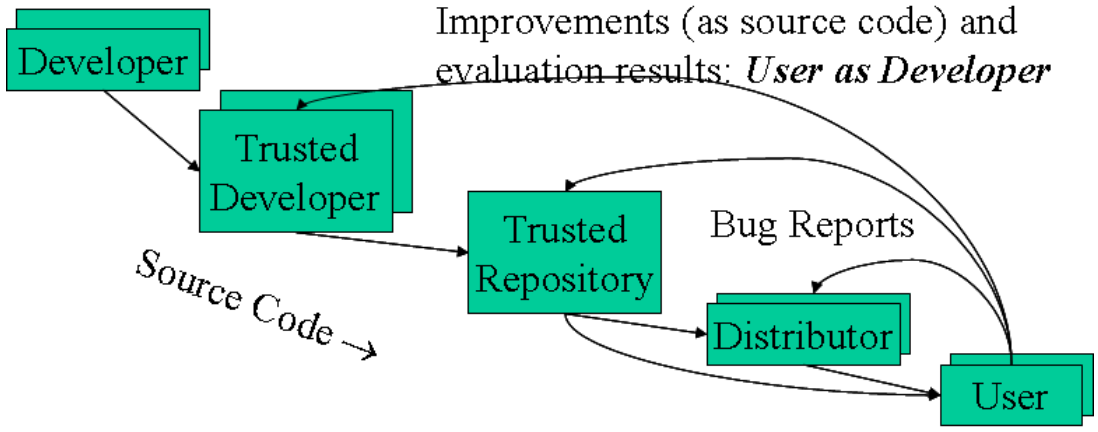


Figure 21.1 OSSD Life Cycle

The project management process begins with an idea that is ultimately enhanced and proposed to other developers. Afterward, code is designed and implemented (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011). Once the initial release is finished it is distributed amongst the design community along with the source code for quick debugging.

This initial release is called the Alpha release. The purpose of the release is to complete a peer review for quick feedback. This cycle continues until the development community is satisfied with the final product.

The code is finally submitted to the project lead for review. This person(s) is recognized as the project maintainer and their responsibility is to determine whether the code should be accepted into the code tree or not. Complex projects may have multiple layers of maintainers depending on the project requirements. In this instance, once the code has been approved by all maintainers it is included into the main source tree for publication (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

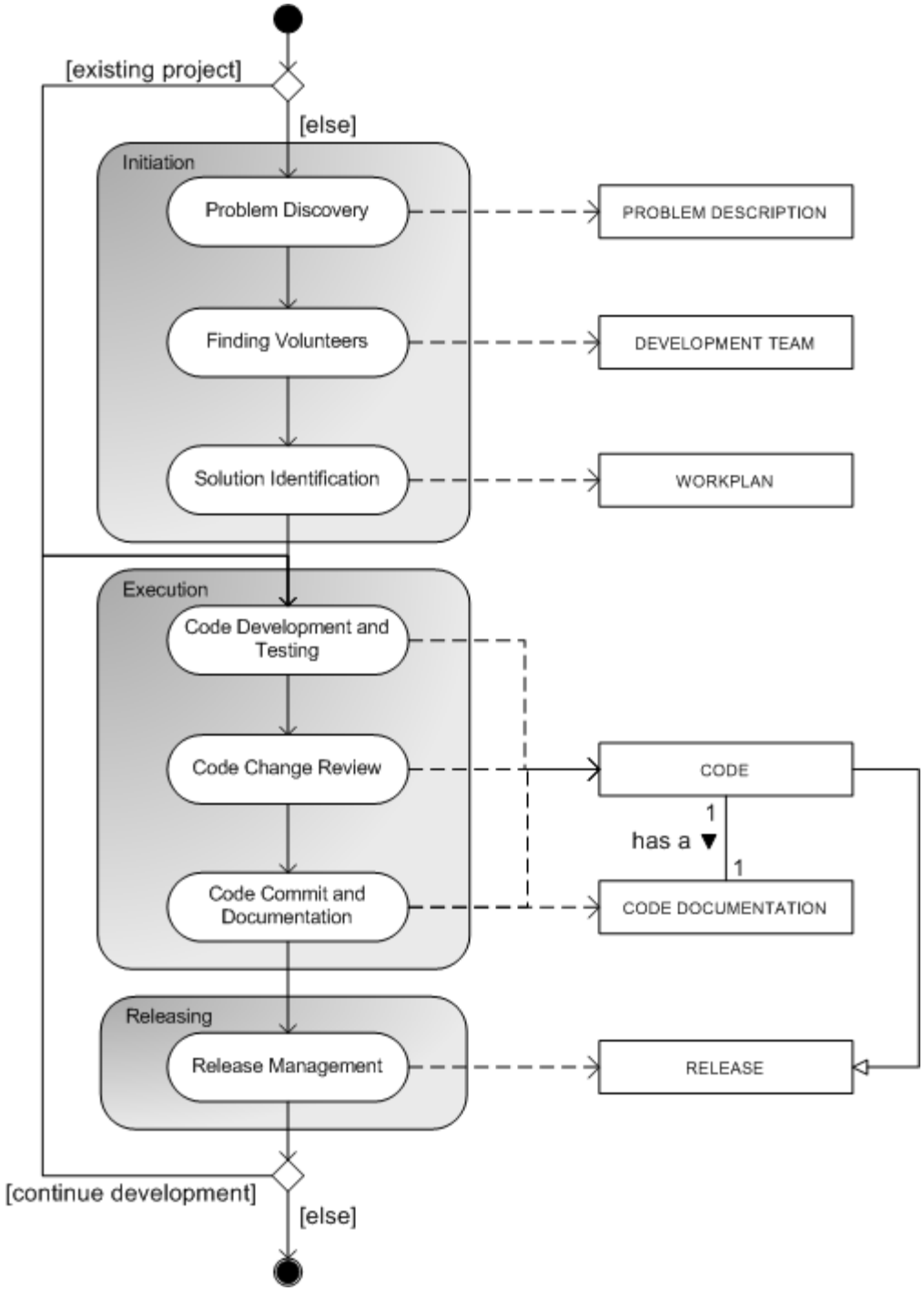


Figure 21.2 OSSD Project Flow

The Open Source Software Development Life Cycle

With traditional development methodologies the requirements are explicitly stated and a contract or Statement of Work (SOW) is created. Once the project requirements have been finalized, the project manager ensures they are delivered via the project management process while maintaining cost, scope, and schedule. The open source process differs from

traditional design methodologies in many ways that are discussed below (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Feature Request Process

Feature requests should be known to every member of the team. They are prioritized and broadcasted using a process known to everyone. This ensures common understanding among the team members, significance of requested features, and their development urgency. The process begins with the proposal sent to a mailing list. The list is made by whoever is willing to lead the team. Since members are in different geographical locations, a common medium is used to communicate and discuss the proposal.

The purpose of the proposal is to notify members of the requirements and gather feedback and acceptance before starting the project (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011). Project contributors and maintainers evaluate the proposal to see whether it is fitting for future increments or not (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011). If the request is approved, the goals and targets are set and the development process begins.

Architectural and Design Discussion

One of the major contributing factors in the open source development process is the level of transparency present. The process is open to anyone interested in it and it enhances the level of collaboration among the members. Communication is vital as members are from different areas. A communication medium that is acceptable to all is used for discussions, decision making, and code review (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

The most common communication channel used is a mailing list because it is transparent, self-documented, and allows anyone in the project development team to participate. This also includes stakeholders and end users who may be monitoring the list to understand the requirements and provide feedback (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Besides the mailing list, via internet live sessions and discussions are held, usually in English, to aid the design process. Electronic media is used for this purpose (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Collaboration on Implementation

The open source development process places great emphasis on collaboration and peer review from beginning till end. Since with open source development the teams rarely meet each other, communicating their plans and tasks clearly helps other members with their piece of the project. The code developed is compatible with any environment making it easy for individual developers to submit their code and get it evaluated by peers (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Source Code Submission

Submission of code in the open source development process is incremental in nature. A set of developers collaboratively begin the code design process. When the code is functional and ready to be delivered it is submitted to the project mailing list. Stakeholders provide feedback and set the acceptance and rejection criteria.

In the case of rejection, the developers will revise their code and resubmit it in the next release. The code is in smaller batches because it makes it easier to review, test, and redistribute. Also submitters are encouraged to submit their code in smaller increments to aid the review process. Smaller batches are rarely the cause of any disastrous effect on the project as it is easy to identify bugs and errors instantly.

The acceptance of code results in it being included in the main development tree. However, in a large project the code may have to go through more than one maintainer to be included in the main development tree (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Continuous Testing and Integration

The work in open source development is highly distributed and the developers are geographically dispersed from each other. This is why frequent incremental development is encouraged to identify errors earlier and fix them as soon as possible. When delivering a large project the team can ask for code submittal after only a week or 10 days. This speeds up the development process and increases efficiency.

Additionally, some code may also be developed to be tested directly by the end user. Feedback is gathered and taken into consideration while developing the next release. Open source ensures that the code doesn’t have a major impact on the overall project and identifies

bugs and fixes them as soon as possible to smooth out the design process (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Source Code Release

As mentioned, usually open source development projects are not perceived to be reliable. However, in actuality, with few exceptions, projects developed using open source are stable and open to alteration and suggestions. This ensures that the project is kept up-to-date and the latest versions of the effort are released frequently. The process of code release varies from project to project but a team is nominated to monitor the quality assurance metrics (QA metrics). When the release criterion is met the team releases the software to be used by the general public and evaluate the performance with the help of feedback (Ph.D., Ibrahim Haddad; Warner, Brian, November 2011).

Enhancement Process in OSS projects

Enhancements can be described as adding additional features to an already existing product to augment its value in the market place. The enhancement process enables the software design team to identify any possible improvements to keep the product competitive. Usually the enhancement cost is, on average, 40% of the original development cost (Aurum, Ghapanchi, & Aybuke).

In traditional development any changes or enhancements are made through the maintenance performer. He is responsible for maintaining, enhancing, correcting, and retargeting. A user wanting any changes submits a report and requests changes to be made to certain aspects of the product. The maintenance performer will accommodate these changes only if he deems it necessary. However, in open source development the users are able to make these alterations on their own without depending on anyone else (Aurum, Ghapanchi, & Aybuke).

The continuous enhancement process is an aspect that makes open source development different from traditional development processes. The “Adding Features” process is a function of the peer review and bug fixing operations. Additionally, open source development incorporates “release management” that is responsible for the handling and release of new revisions. Furthermore, “Feature Requests” by the users are handled through a planned tracking system that provides a complete infrastructure for assessing the feature, developing it, and finally implementing, it (Aurum, Ghapanchi, & Aybuke).

Additional features are developed more rapidly in an open source environment when compared to closed source development because OSS supports creativity in all forms. Therefore, one could make the argument that adding any additional features in a closed software process is more tiresome and time-consuming when compared to open source development (Aurum, Ghapanchi, & Aybuke).

Researchers believe that the success of open source projects depends greatly on the enhancement suggestions received and implemented by the developer. This indicates that the public is actually taking an interest in the software. In fact, the feedback at times becomes additional features.

Admittedly, there has been more research to assess the effectiveness of an enhanced process for closed software development versus the little research that has been conducted for Open Source Software Development. However, some suggest the effectiveness can be measured by the amount of feedback received and by the number of revisions released (Aurum, Ghapanchi, & Aybuke).

Phases of the Open Source Software Development Process

The open source development process has been referred to as a bazaar previously. It tells us that there is no sequence for anything and anyone is allowed to do whatever they will. It is hard for developers to come up with predefined phases for open source, but below is listed what are believed to be applicable (Myers, 2009).

- **Phase 1 - Concept:** Before developing anything, a new concept is required. It, of course, starts with a vision to work on that later becomes documented. If a team is working on the project, the idea is communicated to them using the chosen channel of communication. However, open source doesn't include a lot of documentation. Just the main scope of the project is documented for future reference (Myers, 2009).

- **Phase 2 - Bootstrap:** In this phase, there are just one or two supervisors to govern the project. The developers may post the progress on a blog or write articles about the progress, other than this no further documentation is involved (Myers, 2009).

- **Phase 3 - Early Development:** The developmental process is far quicker compared to traditional development methodologies. Increments are released frequently and the user starts their testing instantly. They later provide feedback for improvements. Some suggestions are documented and implemented, while others may be ignored (Myers, 2009).

- **Phase 4 - Early Adoption:** Open source projects are easily adopted by many organizations because there is great potential for improvement. Furthermore, multiple developers can work on a single idea. This gives the user an option to pick from and implement it within their organization (Myers, 2009).

- **Phase 5 - Development:** With the increase in demand for more innovative products the software development industry is advancing rapidly. People are continually submitting batches of code for testing to validate whether it will be successful or not. The documentation is moderate at this stage, but, of course, some projects may require more documentation than others (Myers, 2009).

- **Phase 6 - Adoption:** With every passing day more and more talented developers are stepping into the software development process through open source. The software developed is feasible for any environment and is adopted by many users (Myers, 2009).

- **Phase 7 - Maturity:** There are often several batches of code that are being released at any given time and they all require instant debugging. After reaching its maturity level, the code is either made available for use or it is sent back to be revised (Myers, 2009).

- **Phase 8 - Mainstream:** The open source development process is becoming increasingly popular day by day. Developers have made their own websites to promote their software and it is now a mainstream process used widely around the globe.

Advantages of Open Source Software Development

The advantages of the open source development methodology are listed below (Rosenblum, 2012).

- The development methodology is free to use and costs are insignificant.

- Any developer willing to develop software with a low budget can opt for the open source development process. It is easy to create, modify, and distribute open source software.
- Open source development is relatively secure since code is readily available for use and developers can actively test and review for intentional code inaccuracies or bugs.
- Users do not have to sit around and wait for the next release of a software product. They can engage in the development process and resolve any errors and fix bugs themselves.
- Open source software is not necessarily dependent on the authority that created it. Even if the company becomes obscure, the code can still be used by many developers to enhance a product's value.
- There are no rigid rules and regulations that must be followed at all times and paperwork is kept to a minimum.
- Companies using open source software do not usually have to worry about licensing or serial number activation issues.

Disadvantages of Open Source Software Development

The disadvantages of the open source development methodology are listed below (Rosenblum, 2012).

- One major disadvantage is that the process is not easy to use or something that can be learned in a single day.
- Often, users have to spend a lot of time learning just how, exactly, the software product works. For example, Linux is not an easy operating system to learn for many people.
- A user learning a new open source product may have to hire a professional for training. This, of course, is an additional expense.

- There are often few applications that run with open source operating systems as well as traditional environments like Windows, which presents developers with additional challenges.
- There are generally several different versions of a product being developed at the same time. This can often confuse an end user regarding what feature is present in a version of a release of the product.
- An open source software product may require an additional third party software product.
- The decision to use the open source development process purely for cost cutting reasons usually results in disastrous endings. A full and thorough analysis should be performed before making any decision of this magnitude.

Conclusion

Well, this is the end of the journey and the in-depth look at the different project management life cycle options that can be chosen to efficiently complete a project. Each of these chapters has strived to cover the history of each life cycle model, along with its detailed usage. Additionally, both the advantages and disadvantages of using each life cycle has been detailed at the end of each chapter. All of this was not to make the practitioner an expert in any one life cycle, but to give them the knowledge necessary to select the right project management life cycle for success. Below, in this Conclusion, is provided some parting thoughts.

Project Management's Challenging Environment

Gone are the days when projects were separate activities with dedicated resources. Today's world is a dynamic one and many people are forced to work on multiple projects at a time. This requires extensive project management skills from all team members regardless of their key duties. Whether it is the project manager focused on the overall management of the effort or a skilled professional working on specialized tasks, today's projects demand everyone to work in an interdependent fashion that is best for the success of the deliverable.

Life Cycle Selection

Having reviewed all the life cycles available, it is hopefully clear just how important the life cycle selection is to your project. A life cycle model is a two edge sword. It has the ability to be the basis for both project success and failure.

Consider for a moment a project to build a residential home. This is a project that occurs on a regular basis in an industry surrounded by best practices. However, the project would be a complete failure if you were to use the Prototype Model because nobody prototypes homes for obvious reasons. Equally, it would be a mistake to use a software project management life cycle that focused on ambiguous requirements, speed of execution, and little documentation. This is a project requiring strict adherence to stages and thorough documentation.

It is crucial to evaluate all aspects of a project management life cycle as it relates to a project or projects and the business environment. Once an approach is selected, the project manager has the duty to review the basis of the decision again with management so that it is agreed that the right model has been selected. Selecting the wrong life cycle not only leads to

failure but also wasted resources. This decision does not need to be completed for all projects if they are all the same kind of project in the same department and industry. Instead, the decision can be made at the project management office (PMO) level and reviewed and approved by executive management.

After the appropriate life cycle is selected, it is important to tailor it to the individual project. By definition, each project is a unique effort. As such, each has its own requirements and the approach may need to be altered to adjust for idiosyncrasies. Today's projects are dynamic and changes are often fluid. The project manager and the team should always be prepared to accommodate reasonable changes during the course of the effort.

Keep it Simple – Triple Constraints

Projects can get complex. In order for them to be managed properly the project manager, team, and executive management must keep their eye on the triple constraints. As politics come into play, technical issues become overwhelming, and stakeholders get frustrated, there has to be a yard stick to measure the project by. This yard stick is and always will be the triple constraints of project management and they will never change.

The triple constraints of project management are quite simply scope, time, and resources and quality is merely a function of the triple constraints. Quality is not delivering more than what was promised or required. That is scope creep. Quality is delivering the required scope on time with the resources allocated. The triple constraints can seem simple and boring and they are. However, most project failures can be traced back to failure to manage the triple constraints of the effort. When faced with any difficult decision on a project, the solutions must always be evaluated against the triple constraints.

Project Management

Projects are how things get done. That is all there is to it. Even before we called them projects and this focus on project management, tasks always had a finite beginning and a finite end. However, failures cost money. In fact, in almost any given year the monies lost due to challenged and failed IT projects alone in the USA is approximately 80 billion dollars. This highlights the reason to start with the end in mind by choosing the right project management life cycle.

Today, depending on what report is quoted, the success rate of all projects is only approximately 30%-50% in any given year. A company with an above average success rate

will create a better corporate image than their competitors and have the opportunity for increased profit margins. The same is also true for an individual that works in project environments and is able to set themselves apart from the rest of their professional community.

Just the Beginning

Successful execution of a project goes much deeper than just selecting the right life cycle though. It takes a thorough understanding of the key aspects of the effort, among other things, to assist in processing requirements for the purpose of success (Hanif). However, once an approach has been selected, available resources must then be committed to the project and managed accordingly.

During the execution phases of the project all of the planning in the previous phases must be put into action instead of discarded in order for the effort to reach a satisfactory end. Finally, when the last phases of a project arrive it is necessary to ensure that all requirements have been completely fulfilled and the project is closed formally. If all phases of the project were executed properly up to this point and documented accordingly, then it should be a fairly simple task to close the project out and perform post mortem activities.

Project Life Cycle Summary

While this book evaluates 24 different project management life cycles to give the reader the knowledge to choose the correct approach for their endeavor, the table below summarizes each life cycle for ease of selection. The chart below can be used for quick selection, but each chapter should be reviewed before final decision.

Project Life Cycle	Use For
Rational Unified Process (RUP)	Best suited for large scale projects and joint ventures. Effective when agreement between all stakeholders is important for completion of a project. Feedback is considered at all stages.
PRINCE2	Useful with all project types regardless of their scale. Details step-by-step processes and guidelines. Consistent approach. Relies on constant client involvement. Can be improved over time. Works with specified roles.
System Development Life Cycle Model (SDLC)	Contains interdependent activities. Used when requirements are clear. Effective when company has expertise and experience in dealing with similar projects.
Capability Maturity Model (CMM)	Well suited in situations where improvement in processes is desired. Effective for projects where quality is the focus. Based on actual practices. Provides all required documentation. Publicly available for use by anyone.

TenStep	Useful for projects of all kinds. Step-by-step approach. Can be modified to suit needs of a project. Designed to be flexible. Provides a scalable approach.
Agile: Extreme Programming (XP)	Light weight methodology. Relies on defined deliverables. Able to manage technical and schedule risks. Needs integrated team and customer involvement. Effective when time to market is short. Favors application development over documentation.
Agile: Scrum	Completes projects in small sprints. Light weight methodology. Needs fast decisions and empowered project teams. Able to manage schedule and technical risks. Preferred by organizations with active collaboration between teams.
Agile: Crystal	Adaptable to projects of varying size and complexity. Works well with small team size but needs proper communication. Can be tailored for any size team.
Agile: Dynamic Systems Development Method (DSDM)	Oldest Agile methodology. Manages risks, costs, and schedule well. Reduces margin of error. Works well with one or two teams working on the project that assume complete responsibility.
Agile: Lean Development	Light weight methodology. Requires empowered teams. Manages changes well.
Agile: Feature-Driven Development (FDD)	Effective if organization is shifting from phase based to iterative approach. Used by large teams working on projects. Defines tasks and roles clearly.
Rapid Application Development (RAD)	Effective when time to market is short. Facilitates fast development outputs. Inexpensive and efficient. Meets all user requirements. Tends to reduce maintenance expenses. Delivers high quality products.
Unicycle Model	Used where focus on technical aspects is required. Can be molded to the needs of any project size. Effective when risks must be lowered. Delivers ease of implementation and reliability. Easy to apply controls. Tends to be lenient on documentation requirements.
Code-and-Fix Approach	Good for small companies. Ad-hoc approach with no structured process. Used where customer acceptance at each step is not required. Results can be seen immediately. Lenient on processes. Can be used by novice developers.
Scaled Agile Framework (SAFe)	Effective with larger, multi-team projects. Different teams can run different implementations of Agile. Facilitates consistent strategy across departments. Can improve product development lead times.
V-Methodology	Linear development model. Works with defined project requirements. Makes tool selection easier because of availability of clear requirements. Good for teams that work with sequential activities. Preferred mostly for its simplicity. Project cost can easily be estimated. Supports multi-layer design. Facilitates early detection of errors.
Waterfall Model	Oldest SDLC. Best suited for construction and manufacturing niches. Activities need to be completed sequentially. Emphasizes documentation and planning. Relies on agreed requirements. Works well with stable project requirements and defined scope.

Prototype Model	Accommodates projects with ambiguous requirements. Best suited when mockup or prototype is to be presented to client through wireframes or GUI. Able to receive detailed requirements from customers before full scale development is started.
Spiral Model	Iterative in nature. Used when risk avoidance is important. Able to manage technical risks. Can work well with evolving requirements. Relies on risk calculations and possible risk avoidance completed at the beginning of the project.
Synchronize and Stabilize	Best suited for large-scale software systems. Allows independent teams to work on separate parts of the same project at the same time. Debugging is crucial. Specifications are only finished once the whole project has been completed. Known for its use with innovative product developments.
Reverse Engineering Development	Shortens lead times with new product development. Widely accepted in manufacturing, reproduction, and industrial design environments.
Structured System Analysis and Design Method (SSADM)	Used by government entities. Preferred for large-scale information systems. Used in industries with high volume business events. Involves detailed documentation and specifies exact work flow. Reviews at each stage. Improves quality and provides reusability.
PRAMIS	Has a progressive approach and makes forecasting easier. Reduces operating expenses. Provides for simple interpretation. Used in projects where flexibility is required. Financial and accounting focused.
Open Source	Preferred when cost of software licensing wants to be avoided. Involves community feedback during the development process. Code creation and modification is straight forward. Users can fix bugs themselves. Lack of rigid requirements.

Table C.1 Life Cycle Selection (Council, 2013) (Rothman, 2008) (Scheid, 2015)

Closing Thoughts

This book has been meant to deliver an overview of the life cycles available for completing a project, the importance of choosing the right process, and how to select a project management life cycle. It was not meant to provide detailed information on each and every life cycle. For further information on the life cycles included in this book, it is recommended that the reader consult additional resources. Great effort was taken to include both new and old project management methodologies in this book. This was to cover all established and emerging models and give a broad picture of the options available for comparison. It is hoped that this book will serve as a valuable resource to students, project teams, professionals, and the general public alike.

Works Cited

(n.d.). Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=7241

1.0 Define the Work. (n.d.). Retrieved from TenStep You Can Manage: http://www.tensteppm.com/open/1.0DefinetheWork.html

10.0 Manage Procurement. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/10.0ManageProcurement.html

2.0 Build the Schedule and Budget. (n.d.). Retrieved from TenStep You Can Manage: http://www.tensteppm.com/open/2.0BuildSchedandBudget.html

3.0 Manage the Schedule and Budget. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/3.0ManageSchedandBudget.html

4.0 Manage Issues. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/4.0ManageIssues.html

5.0 Manage Scope. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/5.0ManageScope.html

6.0 Manage Communication. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/6.0ManageComm.html

9.0 Manage Quality and Metrics. (n.d.). Retrieved from TenStep You Can Change: http://www.tensteppm.com/open/9.0ManageQualityandMetrics.html

A SRE METHODOLY FOR MODERNIZATION. (n.d.). *IJAET*.

Accountants, C. I. (n.d.). *Chartered Institute of Management Accountants*. Retrieved from http://www.cimaglobal.com/

Admin. (2013, June 1). *Software Development Life Cycle Models.* Retrieved from SPM Info Blog: http://www.spminfoblog.com/post/72/software-development-life-cycle-models/

Admin, S. (n.d.). *The ScrumMaster Role*. Retrieved from Scrum Methodology: http://scrummethodology.com/the-scrummaster-role/

Advantages and Disadvantages of SDLC. (n.d.). Retrieved from ETERNAL SUNSHINE OF THE IS MIND: http://eternalsunshineoftheismind.wordpress.com/2013/03/03/advantages-and-disadvantages-of-sdlc/

Agarwal, R., Prasad, J., Tanniru, M., & Lynch, J. (2000, November). Risks of rapid application development. *ACM Digital Library*, pp. Volume 43 Issue 11es, Article No. 1 .

Allan, G. (1997). *Project Development Life-Cycle Models.*

Alliance, S. (n.d.). *Core Scrum — Values and roles*. Retrieved from Scrum Alliance: http://www.scrumalliance.org/why-scrum/core-scrum-values-roles

Ambler, S. (n.d.). *Choose the Right Software Method for the Job.* Retrieved from Agile Data: http://www.agiledata.org/essays/differentStrategies.html

Ambler, S. W. (2005). *A Manager's Introduction to The Rational Unified Process (RUP).*

AN INTRODUCTION TO SSADM. (Jan 2012).

Atwal, H. (2008). Retrieved from Capability Maturity Model Disadvantages: http://www.cs.nott.ac.uk/~pszcah/G53QAT/Report08/hsa06u-WebPage/hsa06u-WebPage/disadvantages.html

Aurum, Ghapanchi, A. H., & Aybuke. (n.d.). An Evaluation Criterion for Open Source Software Projects: Enhancement Process Effectiveness.

Averkamp, H. (n.d.). *What is Job Order Costing*. Retrieved from Accounting Coach: http://www.accountingcoach.com/blog/what-is-job-order-costing

Azam, Z. (2010). *Web Projects Management Between Theory and Practical Application.*

Basics, S. (2010, October 10). *The Scrum Team Roles*. Retrieved from Scrum Methodology: http://scrummethodology.com/the-scrum-team-role/

Benefield, P. D. (2007). An Introduction to Aglie Project Management with SCRUM. *The SCRUM Primer*.

Beynon-Davies, P. (2002, July 1). Design breakdowns, scenarios and rapid application development. *Science Direct*, pp. 579–592.

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (Volume 8, Number 3, 1 September 1999). Rapid application development (RAD): an empirical review. *European Journal of Information Systems*, 211-223(13).

Bhattacharyya, D. (2010). *Management Accounting.* ISBN 978-81-317-3178-9.

Boehm, B. (May 1988, May). *A spiral model of software development and enhancement.* IEEE Software.

Boehm, B. W. (1988). *A Spiral Model of Software Development and Enhancement.* TRW Defense Systems Group .

Brown, A. W. (1996). *Component-Based Software Engineering.*

Buehring, S. (2013, July Monday). *The principles of PRINCE2*. Retrieved from Knowledge Train: http://www.knowledgetrain.co.uk/prince2-principles-explained-part-1.php

Capability Maturity Model. (n.d.). 14.

Caplan, D. (2006). *Management Accounting Concepts and Techniques*. Retrieved from Dennis Caplan: http://www.denniscaplan.fatcow.com/TOC.htm

Changede, A. (2013). *Testing - V model.* Retrieved from Career Ride: http://www.careerride.com/testing-v-model.aspx

Charvat, J. (2003). *Project Management Methodologies.* New Jersey: John Wiley and Sons, Inc.

Clair, C. L. (2005). *How to succeed in the enterprise software market.* Idea Group Inc (IGI).

Cockburn, A. (1998-2004). Crystal Clear- A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects. *Human and Technology*.

Cockburn, A. (n.d.). *Crystal methodologies*. Retrieved from Alistair Cockburn: http://alistair.cockburn.us/Crystal+methodologies

Cohn, M. (n.d.). *Topics in Scrum*. Retrieved from Mountain Goat Software: http://www.mountaingoatsoftware.com/agile/scrum/overview

Coleman, G., & Verbruggen, R. (1998). A Quality Software Process for Rapid Application Development. *Software Quality Management VI*, pp. 241-259.

Consultant, C. (2013, June 5). *Advantages and Disadvantages of CMMI-DEV for a Software Development IT Organization? And how to overcome these disadvantages?* Retrieved from CMMI Consultant Blog: http://www.cmmiconsultantblog.com/cmmi-faqs/advantages-and-disadvantages-of-cmmi-dev-for-a-software-development-it-organization-and-how-to-overcome-these-disadvantages/

Correa, H. (n.d.). *Introduction to Scrum*. Retrieved from Code Magazine: http://www.codemag.com/Article/0805051

COTS, S. R. (n.d.). PHASE 7: TEST PHASE.

Council, V. G. (2013, July 1). *Selecting a project management methodology.* Retrieved from PM Guide 01: https://ofti.org/wp-content/uploads/2013/08/PM-GUIDE-01-Project-management-methodology-selection-guideline.pdf

Csaba, P. (2013, Jan 23). *From Scrum to Lean.* Retrieved from Tuts Plus: http://code.tutsplus.com/articles/from-scrum-to-lean--net-29420

Current ITS Projects. (n.d.). Retrieved from Information Technology Services: http://www.utexas.edu/its/projects/framework.php

Custom, S. R. (n.d.). PHASE 3: PLANNING PHASE.

Custom, S. R. (n.d.). PHASE 5: DESIGN PHASE.

Cusumano, M. A., & Selby, R. W. (1998). *Microsoft Secrets.* Free Press; 1st Touchstone Ed edition.

DeBeneditti, J. (n.d.). *The Impact of Management Accounting Techniques on Profitability*. Retrieved from Small Business: http://smallbusiness.chron.com/impact-management-accounting-techniques-profitability-77635.html

DeGrace, P., & Stahl, L. H. (1990). *Wicked Problems, Righteous Solutions: A Catolog of Modern Engineering Paradigms: A Catalogue of Modern Software Engineering Paradigms* . Yourdin Press.

Dip. (2009). *Top 8 Disadvantages of Extreme Programming*. Retrieved from Enterprice Software Blog: http://enterpriseblog.net/a/top-10-disadvantages-of-extreme-programming/

Disadvantages and Advantages of CMMI-DEV. (n.d.). Retrieved from ECCI: http://ecci.com.vn/en/how-to-overcome-disadvantages-of-cmmi-dev-for-a-software-organization-and-convert-them-into-advantages

DSDM. (2012, August 30). Retrieved from Wikipedia: http://s-and-j.co.uk/wiki/index.php?title=DSDM#History

Dubleowseven, P. R. (2014, September 2). *DYNAMIC SYSTEMS DEVELOPMENT METHOD.* Retrieved from Prezi: https://prezi.com/9bjgkx8prep6/dynamic-systems-development-method/

Dynamic systems development method. (2016, May 4). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Dynamic_systems_development_method

Elssamadisy, A. (2013, August 15). *Has SAFe Cracked the Large Agile Adoption Nut?* Retrieved from Info Q: http://www.infoq.com/news/2013/08/safe

Extreme Programming Model. (n.d.). Retrieved from Classle Learning is Social: https://www.classle.net/book/extreme-programming-model#

G, A. (2010, November 23). *Spiral Model Advantages and Disadvantages*. Retrieved from Buzzle: http://www.buzzle.com/articles/spiral-model-advantages-and-disadvantages.html

Giunchiglia, F., & Tomasi, A. (2005). *Software Development Process.* Retrieved from Software Engineering: http://disi.unitn.it/~tomasi/sweng/05/06-Code&Fix.pdf

Goldratt. (n.d.). *Throughput Accounting*. Retrieved from Goldratt: http://www.goldratt.co.uk/resources/throughput_accounting/index.html

Goyal, S. (WS 2007/08). Feature Driven Development : Agile Techniques for Project Management and Software Engineering. Munich: Technical University Munich.

Grady Booch, J. R. (1999). *Unified Modeling Language—User's Guide.* Addison-Wesley.

Hanif, T. (n.d.). *Academia.* Retrieved from Selecting the right project management approach using 6P: http://www.academia.edu/352881/Selecting_the_right_project_management_approach_using_6P

Hansen, D. B. (2001). The Spiral Model as a Tool for Evolutionary Acquisition.

Hardware, M. R. (n.d.). PHASE 4: REQUIREMENTS ANALYSIS PHASE .

Hardware, M. R. (n.d.). PHASE 8: IMPLEMENTATION PHASE.

Hardware, S. R. (n.d.). PHASE 1: INITIATION PHASE .

Hardware, S. R. (n.d.). PHASE 10: DISPOSITION PHASE .

Hardware, S. R. (n.d.). PHASE 2: CONCEPT DEVELOPMENT PHASE.

Hardware, S. R. (n.d.). PHASE 6: DEVELOPMENT PHASE.

Hardware, S. R. (n.d.). PHASE 9: OPERATIONS AND MAINTENANCE PHASE.

Highsmith, J. A. (n.d.). *Agile Software Development Ecosystems.*

History. (n.d.). Retrieved from Crystal Clear Methodology: http://crystalclearmethodology.blogspot.com/2010/05/history.html

History of Unified Process. (n.d.). Retrieved from Enterprise Unified Process: http://www.enterpriseunifiedprocess.com/essays/history.html

Howard, A. (2002, October 10). Rapid Application Development: rough and dirty or value-for-money engineering? *ACM Digital Library*, pp. 27-29 .

Hurst, J. (n.d.). The Capability Maturity Model and Its Applications. *SANS Software Security site*.

Implementing Scaled Agile Framework (SAFe) with VersionOne. (n.d.). Retrieved from Version One: http://www.versionone.com/scale-agile-with-SAFe/

Institute, P. M. (2008). *A GUIDE TO THE PROJECT MANAGEMENT BODY OF KNOWLEDGE (PMBOK® Guide).* Project Management Institute. .

Jailia, E., Mrs.Sujata, Jailia, M., & Agarwal, M. (July, 2011). LEAN SOFTWARE DEVELOPMENT ("As a Survival Tool in Recession"). *International Journal of Software Engineering and Its Applications*, Vol. 5 No. 3.

Janssen, C. (n.d.). *Rapid Application Development (RAD).* Retrieved from Techopedia: http://www.techopedia.com/definition/3982/rapid-application-development-rad

Janssen, C. (n.d.). *Synchronize and Stabilize.* Retrieved from Techopedia: http://www.techopedia.com/definition/26121/synchronize-and-stabilize

Jayaswal, B. K., & Patton, P. C. (2006, September 22). *Software Development Methodology Today.* Retrieved from Inform IT: http://www.informit.com/articles/article.aspx?p=605374&seqNum=2

Kalyankar, L. M. (n.d.). REVERSE ENGINEERING OF SOFTWARE AND TYPES HAZARDS. *JECET*.

Käufl, M. A. (n.d.). Reverse Engineering Technologies for Remanufacturing of Automotive Systems Communicating via CAN Bus. *Journal Of Remanufacturing*.

Kerr, A. (n.d.). *PRINCE2, 7 Principles, 7 Themes, 7 Processes*. Retrieved from Alexander Kerr: http://www.alexanderkerr.co.uk/project-management/prince2-7-principles-7-themes-7-processes

Kessel, C. (2013, January 14). Software History: Waterfall – The Process That Wasn't Meant To Be.

Krishnamurthy, V. (2012, October 15). *A Brief History of Scrum*. Retrieved from TECHWELL: http://www.techwell.com/2012/10/brief-history-scrum

Kruchten, P. (2000). *The Rational Unified Process, An Introduction, Second Edition,.* Addison-Wesley Professional.

leffingwell, d. (2010). *agile software requirements.*

Leffingwell, D. (2014). *What is this framework thing?* Retrieved from Scaled Agile Framework: http://scaledagileframework.com/

Limited, A. (2013). PRINCE2® Maturity Model (P2MM). *Procurement | Programmes & Projects Version 2.1*.

M. Sokovic, J. (2005). RE (reverse engineering) as necessary phase by rapid product development. *Journal of materials processing technology*.

maheshwari, M. S., & Jain, P. C. (2012). A Comparative Analysis of Different types of Models in Software Development Life Cycle. *International Journal of Advanced Research in Computer Science and Software Engineering*.

Malik, A. (n.d.). *How to conduct the Daily Stand-up / Daily Scrum meeting effectively?* Retrieved from Amit Malik PMP, CSM: http://amitsinghmalik.blogspot.com/?view=classic

Malik, N. S. (2013). Management Accounting: Nature and Scope. *Guru Jambheshwar University of Science and Technology*.

Malik, S., & Palencia, J. R. (1999). *Synchronize and Stabilize vs. Open Source.* Ottawa, Ontario, Canada: Carleton University.

Marc Clifton, J. D. (n.d.). 21 Jul 2003.

Martel, F. M. (2002). Extreme Programming Rapid Development for Web bases Applications.

Martin Wieczorek, D. M. (2001). *Software Quality: State of the Art in Management, Testing, and Tools.*

Martin, J. (1991). *Rapid Application Development.* Indianapolis, IN: Macmillan Publishing Co., Inc.

MaryLand. (n.d.). Roles and Responsibilities.

McConnell, S. (1996, July 4). Daily Build and Smoke Test. *IEEE Software*, p. Vol. 13.

McConnell, S. (2010). *Rapid Development: Taming Wild Software Schedules.* O'Reilly Media Inc.

Metafuse, I. (2014). *5 Basic Phases of Project Management*. Retrieved from Project Insight : http://www.projectinsight.net/project-management-basics/basic-project-management-phases.aspx

Methodology, S. (2013, Spetember 19). *Scrum Product Owner*. Retrieved from Scrum Methodology: http://scrummethodology.com/scrum-product-owner/

Moksony, R. (2014, September 16). *Capability Maturity Model Integration (CMMI) Achievement Quick and Lean with codeBeamer!* Retrieved from Intland Software: http://intland.com/blog/alm/capability-maturity-model-integration-cmmi-achievement-quick-and-lean-with-codebeamer/

Monden, Y. (1998). *Toyota Production System, An Integrated Approach to Just-In-Time.* Norcross, GA: Engineering & Management Press.

Morgan, J. (n.d.). *Applying Lean Principles to Product Development.* Retrieved from SAE International: http://www.sae.org/manufacturing/lean/column/leanfeb02.htm

Munassar, N. M., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. *IJCSI International Journal of Computer Science; Vol. 7, Issue 5*, 94-101.

Munassar, N. M., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering . *International Journal of Computer Science* .

Myers, C. (2009). *Lifecycle of an open source project*. Retrieved from Lost Techies: http://lostechies.com/chadmyers/2009/05/28/lifecycle-of-an-open-source-project/

Najam Shahid, O. A. (n.d.). Rational Unified Process.

Nallasenapathi, P. (2006, March). *Lean Development – A team approach to Software Application Development.* Retrieved from SAK Soft: http://saksoft.com/PDF/White_Papers/info_lean.pdf

Nayab, N. (2010). *Advantages of Extreme Programming*. Retrieved from Bright Hub PM: http://www.brighthubpm.com/methods-strategies/87839-advantages-of-extreme-programming/

NEW FRONTIERS OF REVERSE ENGINEERING. (2007). *FOSE*.

Novakouski, M. (n.d.). Summary of Spiral Model.

Osterweil, L. J. (2011). A Process Programmer Looks at the Spiral Model: A Tribute to the Deep Insights of Barry W. Boehm. *International Journal of Software and Informatics, ISSN 1673-7288*.

Palmer, S. (2009, November 20). *An Introduction to Feature-Driven Development.* Retrieved from D Zonw Agile: http://agile.dzone.com/articles/introduction-feature-driven

Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development .* Prentice Hall.

Paper, R. S. (2001). *Rational Unified Process Best Practices for Software Development Teams.*

Paul, D. J. (1997-1999). *Software Life Cycle Models.* Retrieved from Jody Paul: http://www.jodypaul.com/swe/lcm/index.html

Paulk, M. C. (n.d.). A History of the Capability Maturity Model for Software.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1997). The Capability Maturity Model for Software.

Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M. B., & Bush, M. (1993). Key Practices of the Capability Maturity Model Version 1.1. *Carnegie Mellon University Research Showcase*.

Peeters, S. (2013, December 5). *Applying Lean Thinking to Software Development.* Retrieved from Info Q: http://www.infoq.com/articles/applying-lean-thinking-to-software-development

Ph.D., Ibrahim Haddad; Warner, Brian. (November 2011). Understanding the Open Source Development Model. *The Linux Foundation*.

Pincemaille, C. (2008). PRINCE 2: a methodology for project management.

Pinna, R. (2013, May 16). *41 Things You Need to Know about the Scaled Agile Framework® (SAFe).* Retrieved from Rally Dev: http://www.rallydev.com/community/agile/scaled-agile-framework%C2%AE-41-things-you-need-know-about-safe

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit.* Addison-Wesley Professional.

Prehn, S. (2007). Open Source Software Development Process.

PRINCE2 Methodology. (2008). Retrieved from Cupe Projects: http://www.cupe.co.uk/prince2-methodology.html

PRINCE2 Processes. (n.d.). Retrieved from Prince2.com: http://www.prince2.com/prince2-processes

PRINCE2 Project Methodology. (n.d.). Retrieved from tutorials point: http://www.tutorialspoint.com/management_concepts/prince2_project_methodology.htm

PRINCE2™ Project Management Methodology Overview and History. (n.d.). Retrieved from Project Performance International: http://www.ppi-int.com/prince2/prince2-history.php

Principles & Method. (2009). Retrieved from online PRINCE2: http://www.onlineprince2.com/p2nlcom/About-PRINCE2/PRINCE2-Principles-Method

Prototype model of SDLC. (n.d.). Retrieved from Subject Coach: https://www.subjectcoach.com/tutorials/detail/contents/introduction-to-software-development-life-cycle-sdlc/chapter/prototype-model-of-sdlc

Reddy, S. (2013, July 11). *V Model Final.* Retrieved from Slide Share: http://www.slideshare.net/suhasreddy1/v-model-final

Reverse Engineering. (2016, June 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Reverse_engineering#Reverse_engineering_of_software

Riley, L. T. (n.d.). *Introduction: Open Source*. Retrieved from Open Source and Academia: http://www2.bgsu.edu/departments/english/cconline/tayloriley/intro.html

RODRGUES, I. (n.d.). Guidelines for Reverse Engineering Process Modeling .

Ropa, S. (2011, February 15). *Extreme Programming Values – Communication*. Retrieved from Version One- Agile Made easier: http://blogs.versionone.com/agile_management/2011/02/15/extreme-programming-values-communication/

Rosenblum, J. (2012). *Advantages And Disadvantages Of Open Source*. Retrieved from Cloud Tweaks: http://cloudtweaks.com/2012/08/advantages-and-disadvantages-of-open-source/

Rothman, J. (2008, January 1). *Rothman Consulting Group Incorporated.* Retrieved from What Lifecycle? Selecting the Right Model for Your Project: http://www.jrothman.com/articles/2008/01/what-lifecycle-selecting-the-right-model-for-your-project/

Rounds, R. (2008, June 19). *Prototyping History and Prototype Development Information.* Retrieved from Articles Factory: http://www.articlesfactory.com/articles/hobbies/prototyping-history-and-prototype-development-information.html

Rouse, M. (2005, September). Synchronize-and-stabilize (sync-and-stabilize).

Royce, D. W. (1970). Managing the Development of Large Software Systems.

Sabale, R. G., & Dani, D. A. (2012). Comparative Study of Prototype Model For Software Engineering With System Development Life Cycle. *Journal of Engineering* .

Saddington, P. (2014, February 25). *The Scaled Agile Framework (SAFe) – A Review.* Retrieved from Agile Scout: http://agilescout.com/scaled-agile-framework-safe-review/

Saini, M., & Kaur, K. (2014). A Review of Open Source Software Development Life Cycle Models. *International Journal of Software Engineering and Its Applications*.

Satalkar, B. (2011). *Comparison Between Waterfall Model and Spiral Model*. Retrieved from Buzzle: http://www.buzzle.com/articles/comparison-between-waterfall-model-and-spiral-model.html

Sathiparsad, N. (2003, January). Synchronize and Stabilize: A framework for best practices. South Africa.

Scacchi, W. (2001). Process Models in Software Engineering. *Encyclopedia of Software Engineering*.

Scaled Agile Framework. (2016, June 21). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Scaled_Agile_Framework

Schach, S. (2007). *Software Engineering.* New York: McGraw Hill.

Scheid, J. (2015, July 22). *Bright Hub Project Management.* Retrieved from Project Management Methodologies: How Do They Compare?: http://www.brighthubpm.com/methods-strategies/67087-project-management-methodologies-how-do-they-compare/

Schwaber, K. (n.d.). SCRUM Development Process.

Schwartz, M. (2001, November 12). *Reverse-Engineering.* Retrieved from Computer World: http://www.computerworld.com/article/2585652/app-development/reverse-engineering.html

serena.com. (2007). An Introduction to Agile Software Development. *SERENA*.

Service, b. M. (2007). PRINCE2 Topic Gateway Series No. 23. *CIMA*.

Shah, B. (2008, September 24). *Spiral Model*. Retrieved from System Analysis and Design: http://systemanalysisanddesign.blogspot.com/2008/09/spiral-model.html

SidKemp. (n.d.). *The History and Purpose of the Capability Maturity Model (CMM)*. Retrieved from Hubpahes: http://hubpages.com/hub/The-History-and-Purpose-of-the-Capability-Maturity-Model-CMM

Singh, A. (2012). Classification of Software Projects On the Basis Of Its Features and Then Suggesting Appropriate Life Cycle Model Based on the Features. Curtin University.

singh, b. (2010, December 29). *The WINWIN Spiral Model*. Retrieved from Software Engineering: http://soft-engineering.blogspot.com/2010/12/winwin-spiral-model.html

Spiral Model. (n.d.). Retrieved from One Stop Testing: http://www.onestoptesting.com/sdlc-models/spiral-model.asp

Srinath, R. (2009, May 11). *V - Lifecycle Methodology.* Retrieved from City Tech: http://www.citytechinc.com/us/en/blog/2009/05/v_-_lifecycle_method.html

Stafford, P. (2003). *Software Maintenance As Part of the Software Life Cycle.* Boston: Department of Computer Science, Tufts University.

Stanleigh, M. (n.d.). *Process Management vs Project Management*. Retrieved from Business Improvement Architects: http://www.bia.ca/articles/pj--pm-vs-pjm.htm

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice.*

Stein, S. (2004, August 30). *Process Model, Code and Fix.* Retrieved from http://emergenz.hpfsc.de/html/node41.html

Structured Systems Analysis and Design Methodology. (n.d.). *ITC InfoTech Solutions*.

Sutherland, K. S. (2010). SCRUM.

System Development Life Cycle. (2010). Retrieved from blog.37: http://www.blog37.com/blog/2010/07/system-development-life-cycle/

Systems Development Life Cycle Volume 1 Introduction to the SDLC. (2006).

Takang, A. A., & Grubb, P. (2003). *Software Maintenance: Concepts and Practice.* Singapore: World Scientific Publishing Company.

TenStep, I. (2011). *TenStep® Project Management Process Summary.* Retrieved from Ten Step - You Can Manage: http://www.tenstep.com/open/miscdocs/TenStepSummary.pdf

The 7 PRINCE2® Principles: A Closer Look. (2010). Retrieved from Project Management: http://project-management.learningtree.com/2010/04/03/the-7-prince2-principles-a-closer-look/

The Four XP Values. (n.d.). Retrieved from Software Reality : http://www.softwarereality.com/lifecycle/xp/four_values.jsp

The Horror Of The Scaled Agile Framework. (2012, March 21). Retrieved from Neil Killick: http://neilkillick.com/2012/03/21/the-horror-of-the-scaled-agile-framework/

The Spiral Model. (n.d.). Retrieved from Software Engineering: http://courses.cs.vt.edu/~csonline/SE/Lessons/Spiral/index.html

Thibaut, C. (2013, May 13). *Code and Fix.* Retrieved from http://c2.com/cgi/wiki?CodeAndFix

Thomas Kwasa, D. V. (n.d.). A History Of Structured Systems Analysis & Design Methodologies.

Topics in Scrum. (2012). Retrieved from Mountain Goat Software: http://www.mountaingoatsoftware.com/agile/scrum/product-owner

University, M. (n.d.). *PROJECTMANAGEMENT METHODOLOGY FRAMEWORK.*

Vajda, J. P. (2010, May 3). *Lean Software Development Principlesq.* Retrieved from Slide Share: http://www.slideshare.net/jpvajda/lean-software-development-principles

Viceconti, M., Zannoni, C., Testi, D., Petrone, M., Perticoni, S., Quadrani, P., . . . Clapworthy, G. (2007, February). The multimod application framework: A rapid application development tool for computer aided medicine. *Computer Methods and Programs in Biomedicine*, pp. Volume 85, Pages 138–151.

Vitez, O. (2014). *What is Project Management Accounting*. Retrieved from WiseGeek: http://www.wisegeek.com/what-is-project-management-accounting.htm

V-Model. (2016, April 15). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/V-Model

Webster, M. (n.d.). *Successful Change Management - Kotter's 8-Step Change Model.* Retrieved from Leadership Thoughts: http://www.leadershipthoughts.com/kotters-8-step-change-model/#Implementing_and_Sustaining_Change

Wells, D. (1999, 2009). *Simplicity is the Key*. Retrieved from Extreme Programming : http://www.extremeprogramming.org/rules/simple.html

Wesley, A. (2002). *Building J2EE Application with the Rational Unified Process.*

West, D. (2002). *Planning A Project Wth Rational Unified Process.*

What is CMMI? Explain the advantages of implementing CMMI - CMMI. (n.d.). Retrieved from Career Ride: http://www.careerride.com/CMMI-advantages-of-implementing.aspx

What is SCRUM Development? (n.d.). Retrieved from Select Business Solutions: http://www.selectbs.com/process-maturity/what-is-scrum-development

What is Spiral model- advantages, disadvantages and when to use it? (2012, January). Retrieved from ISTQB EXAM CERTIFICATION: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/

What is V-model - advantages, disadvantages and when to use it? (2012, January 13). Retrieved from ISTQB EXAM CERTIFICATION: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/

Wideman, R. M. (2002). Comparing PRINCE2 with PMBoK®.

Wikipedia. (n.d.). *Prototype.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Prototype

William R. Duncan, D. o. (1996). *A guide to Project Management Body of Knowledge.*

Woods, D. (2010, January 12). *Why Lean And Agile Go Together.* Retrieved from Forbes: http://www.forbes.com/2010/01/11/software-lean-manufacturing-technology-cio-network-agile.html

Yetton, P. (1983). *The relationship among group size.*