CHAPTER 4

Software Development Life Cycle

MAJOR TOPICS

OBJECTIVES

At the completion of this chapter, you will be able to:

- Define software development life cycle.
- Describe the Waterfall life cycle model.
- Describe the V-Shaped life cycle model.
- Describe the Incremental life cycle model.
- Describe the Spiral life cycle model.

PRE-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. What is a software development life cycle?

   .....

   .....

2. What are the four main phases of the general software development life cycle model?

   .....

   .....

.....

.....

INTRODUCTION

Software for business applications, whether it is intended to perform a single task or it is intended for use as a company-wide, integrated system, should be tailored to fit the company's unique needs and goals. In the simplest terms, the software should be capable of performing all the functions necessary to perform a task efficiently. While the software should be inclusive, it should not be unduly cumbersome. Careful attention is required to develop software that is both functional and efficient. Once the software is in use, it must be maintained. In this chapter, we will discuss several processes which can serve as guidelines for software development.

SOFTWARE DEVELOPMENT LIFE CYCLE MODEL

The term software development life cycle model is a way of describing the planning, designing, coding, and testing of a software system, as well as the method in which these steps are implemented. A variety of life cycle models exist, but they all include the same constituent parts. All life cycle models take a project through several primary phases: a requirements-gathering phase, a design phase, a construction or implementation phase, and a testing phase. Figure 4-1 provides a simplified illustration of the general software development life cycle model.

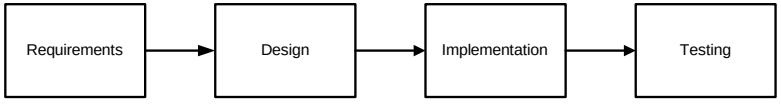

Figure 4-1: General software development life cycle model

Each phase produces feedback that affects the next phase. For instance, the requirements gathered during the requirements phase influence the design, which is translated into working software code during the implementation phase. The software code is verified against the requirements during the testing phase.

Requirements-gathering

During the requirements-gathering phase, the needs of the company are outlined. Managers and users (and in some cases, clients) make their “wish-lists” about what they would like the software to do. Analysts ask questions about the intended use of the software, what type of data will be processed, how the software should handle the data, and how the data can be accessed once in the system.

Following the requirements phase, the software development team should have a detailed list of functions that the system will perform. Emphasis is on the system's goals, rather than the way in which the system will achieve those goals.

Design

In the design phase, the results of the requirements-gathering phase are translated into a software design plan. Focus shifts from the system's results to the way in which those results will be achieved and how the ideas of the requirements-gathering phase are accomplished. Designers consider many different criteria, from the hardware and operating system platform that hosting the software to the way subsystems will communicate with each other.

In essence, during the design phase, the designers attempt to turn the dreams of the managers and users into reality. Emphasis during this phase is on making a practical, working design for what has been outlined in the requirements-gathering phase.

Implementation

In the implementation phase, the results of the design phase are translated into program code. Software that does not meet the needs of the company is wasteful. During this phase the programmers should make it their central goal to fulfill the requirements of the company and to meet the design outlined in the design phase.

The classes and class interactions developed in the design phase are very explicit. They translate directly into the code generated in the implementation phase. Later in this course, design tools will be introduced that actually automate code generation from the output of the design phase.

Testing

In the testing phase, the results of the implementation phase are run through a series of tests to verify that it functions and that it meets goals of the requirements phase. A testing plan is created to describe the unit tests and system tests that will be performed. Unit testing is performed on individual software components. The process of integration brings together all the software components to create a complete system. System testing is performed on the software system as a whole.

WATERFALL LIFE CYCLE MODEL

The Waterfall life cycle model, also known as the classic or linear-sequential life cycle model, is one of the simplest to understand and use. The Waterfall model is characterized by a series of steps that must be completed in a linear, sequential order. Each phase is completed and verified before development progresses to the next phase. Figure 4-2 illustrates the Waterfall life cycle model.

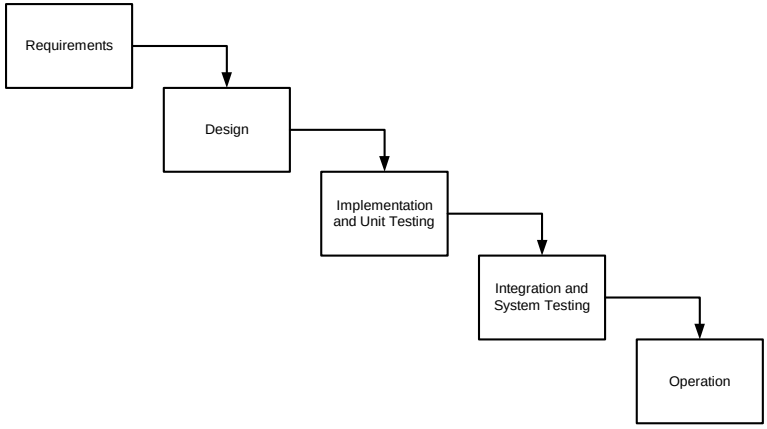

Figure 4-2: Waterfall life cycle model

Following each phase is a formal review process that results in the decision to proceed to the next phase. Testing is performed late in the development process, and phases do not overlap. When one phase is completed, the entire development effort shifts to the next phase. This cascading flow from one phase to another is the basis of the Waterfall model's name.

Advantages of the Waterfall model

As a formalized approach to software development, the Waterfall model is simple and easy to use. This model can be easy to implement and manage because each phase has a specific purpose, and development occurs in only one phase at a time.

The Waterfall model is appropriate for small development projects in which the requirements are well understood.

Disadvantages of the Waterfall model

Although the Waterfall model provides a good introduction to software life cycle models, its usefulness is limited. Due to the rigidity of the model, all requirements must be stated explicitly before development begins. If requirements change or are added, the project must start over from the beginning. No working software is developed until very late in the process, and this delay creates a great deal of uncertainty and risk. If errors are made in requirements-gathering or design, they may not be discovered until very late in the implementation or testing phases.

Due to these issues, the Waterfall model is inappropriate for complex projects. It should not be used for developing object-oriented software, for long-term or ongoing projects, or for projects in which requirements are unknown or subject to change.

V-SHAPED LIFE CYCLE MODEL

Like the Waterfall model, the V-Shaped life cycle model provides a sequential path of discrete phases that must each be completed before development proceeds to the next phase. The phase test plans are developed during each level of development, and once coding is completed, the tests are conducted. Figure 4-3 is a simplified illustration of the V-Shaped life cycle model.

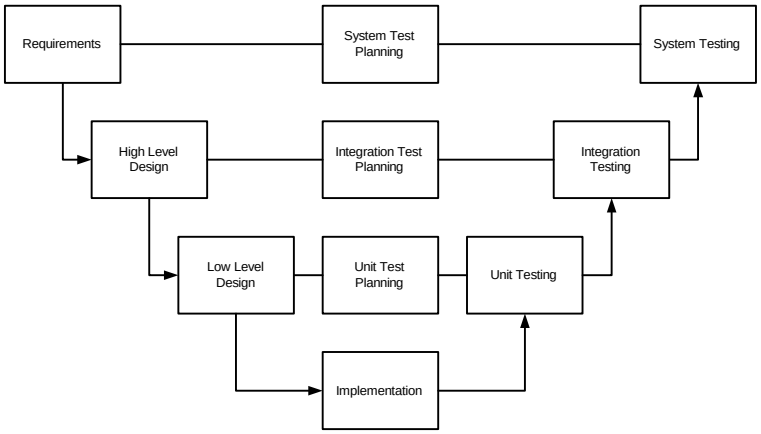

Figure 4-3: V-Shaped life cycle model

The development effort begins with requirements gathering (in the upper-left point of the V). During this phase, requirements are gathered and recorded as they are in the Waterfall model. However, before development moves to the next phase, a system test plan is created. The system test plan will focus on verifying the functionality specified in the requirements definition.

As the V-model moves down the left side, developers focus on high-level system architecture and overall system design. The integration test plan developed in this phase will test the abilities of pieces of the finished software system to operate together. The low-level design phases focus on the design of individual software components, and the corresponding unit test plans developed in this phase are used to test individual components.

Coding is performed at the bottom of the V, during the implementation phase. Once coding is complete, development progresses up the right side of the V, moving through the test plans developed during the earlier phases. If a problem arises during a testing phase, the life-cycle reverts back to its corresponding development phase.

Advantages of the V-Shaped model

Like the Waterfall model, the V-Shaped model is relatively simple and easy to use. Specific goals are defined for each phase. The focus on preparing test plans early in the process gives the V-Shaped model a higher chance for success. Like the Waterfall model, the V-Shaped model is appropriate for small development projects in which requirements are well understood.

Disadvantages of the V-Shaped model

The rigidity of the V-Shaped model makes it inappropriate for most projects. Like the Waterfall model, all requirements must be stated at the beginning of the project, making it difficult to add or change requirements later in the development process. All software development occurs in a single phase, so there are no early working versions or prototypes. The emphasis in test planning appears to reduce risk, but, like the Waterfall model, the V-shaped model risks time and energy in the absence of careful planning. Should the last phase of training, which corresponds to the first phase of development, reveal that a slight modification is necessary, the entire development process must be repeated. Like the Waterfall model, the V-Shaped model is inappropriate for complex projects.

INCREMENTAL LIFE CYCLE MODEL

The Incremental life cycle model builds an iterative approach into the Waterfall model. Development projects are divided into several smaller, more manageable iterations. Each iteration passes through a mini-Waterfall process. Requirement, design, implementation and testing phases are completed for each iteration. Figure 4-4 illustrates the Incremental life cycle model.

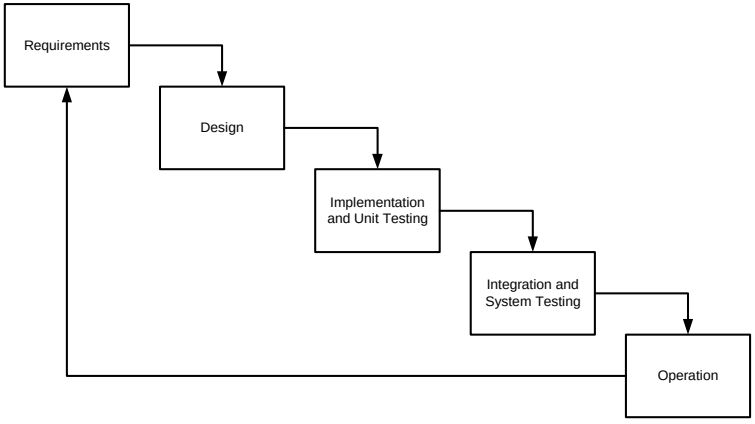

Figure 4-4: Incremental life cycle model

Development begins with requirements gathering and high-level design, and then requirements are divided into iterations. The Incremental model is evolutionary meaning that a working version of the software is created by the end of the first iteration, and subsequent iterations build upon the work of earlier iterations. Low-level design is performed during each iteration, and unit testing evaluates the functionality added during the current iteration. System testing evaluates the way in which the new functionality affects the functionality of the entire system.

Advantages of the Incremental model

Unlike the Waterfall and V-Shaped models, the Incremental model generates a working prototype early in the development process. The iterative nature of the Incremental model makes it more flexible when adding or changing requirements. This model is also easier to test and debug because testing is performed incrementally during each iteration on a relatively small amount of new code.

The Incremental model is an effective tool for managing risk. Chancy portions of development can be identified and carefully managed in their own iteration. In addition to being performed incrementally, each iteration represents a measurable milestone.

Disadvantages of the Incremental model

Like the Waterfall model, the Incremental model exhibits some rigidity in that phases cannot be overlapped. Because requirements-gathering and design are performed during each iteration and not before development begins, projects developed using the Incremental model may incur errors in early iterations. The Incremental model is less risky than the Waterfall or V-Shaped models, but it may be inappropriate for large, long-term projects.

SPIRAL LIFE CYCLE MODEL

The Spiral life cycle model is similar to the Incremental model but incorporates risk analysis. It is divided into four phases: planning, risk analysis, engineering, and evaluation. A project passes through each of these phases in sequence, repeatedly, in a series of iterations called spirals. At the beginning of the development process, critical requirements are identified for the first spiral. Subsequent spirals add functionality to this baseline spiral.

The Spiral model is represented by a spiral passing through four quadrants, which represent the four phases of development. Requirements gathering is performed in the planning phase. During the risk analysis phase, a formal process is undertaken to identify alternative courses of action and their relative risks. A prototype is also developed during this phase. Software is coded and tested during the engineering phase. During the evaluation phase, the customer has an opportunity to evaluate the output before the project proceeds to the next spiral. Notice that the angular component represents the progress in the current spiral, and the radius represents the project cost. Figure 4-5 illustrates the Spiral life cycle model.

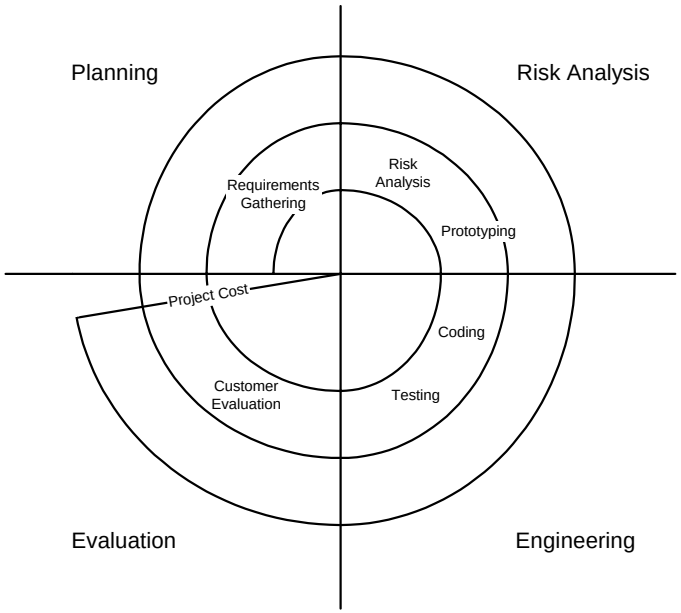

Figure 4-5: Spiral life cycle model

Advantages of the Spiral model

The focus on risk avoidance makes the Spiral model ideal for large-scale and mission-critical products. At its core, the Spiral model is built on earlier software development models, and it borrows from both the Waterfall and Incremental models. Working software code is developed early; thus, the customer is given many opportunities to evaluate the software and plenty of time to ease into adoption of the software.

Disadvantages of the Spiral model

The Spiral model can cost considerably more to implement than other life cycle models. The risk analysis phase requires highly specific expertise, and the project's success depends on the output of this phase. The Spiral model is inappropriate for use in small and medium-scale projects that are not mission-critical.

SUMMARY

All software development life cycle models share common phases of development: gathering of requirements, designing of the software system, coding of software, and the testing of the system. The Waterfall life cycle model is one of the simplest and easiest to use; it consists of five discrete phases that are executed sequentially. The V-Shaped life cycle model adds an emphasis on testing to the Waterfall model. The Incremental life cycle model applies a series of iterations to the Waterfall model. The Spiral life cycle model builds upon the Waterfall and Incremental models and focuses on risk analysis.

POST-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. Why are the Waterfall cycle model and V-shaped cycle model inappropriate for large design projects?

   .....

   .....

2. How does the Spiral model overcome the limitation of the Waterfall cycle model and the V-shaped cycle model?

   .....

   .....

3. Why is the Spiral model only appropriate for large-scale and mission critical applications?

   .....

   .....