Module 2

Software Life Cycle Model

Lesson 4

Prototyping and Spiral Life Cycle Models

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- Explain what a prototype is.
- Explain why and when a prototype needs to be developed during software development.
- Identify the situations in which one would prefer to build a prototype.
- State the activities carried out during each phase of a spiral model.
- Identify circumstances under which spiral model should be used for software development.
- Tailor a development process to a specific project.

Prototype

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- how the screens might look like
- how the user interface would behave
- how the system would produce outputs

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

Examples for prototype model

A prototype of the actual product is preferred in situations such as:

- user requirements are not complete
- technical issues are not clear

Let's see an example for each of the above category.

Example 1: User requirements are not complete

In any application software like billing in a retail shop, accounting in a firm, etc the users of the software are not clear about the different functionalities required. Once they are provided with the prototype implementation, they can try to use it and find out the missing functionalities.

Example 2: Technical issues are not clear

Suppose a project involves writing a compiler and the development team has never written a compiler.

In such a case, the team can consider a simple language, try to build a compiler in order to check the issues that arise in the process and resolve them. After successfully building a small compiler (prototype), they would extend it to one that supports a complete language.

Spiral model

The Spiral model of software development is shown in fig. 2.2. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. 2.2. The following activities are carried out during each phase of a spiral model.

- **First quadrant (Objective Setting)**

  - During the first quadrant, it is needed to identify the objectives of the phase.
  - Examine the risks associated with these objectives.

- **Second Quadrant (Risk Assessment and Reduction)**

  - A detailed analysis is carried out for each identified project risk.
  - Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

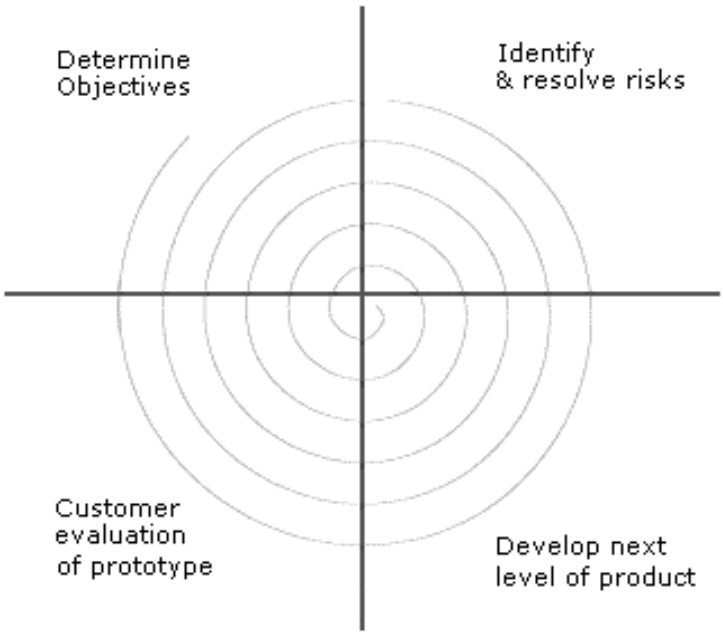


Fig. 2.2: Spiral Model

- **Third Quadrant (Development and Validation)**

  - Develop and validate the next level of the product after resolving the identified risks.

- **Fourth Quadrant (Review and Planning)**

  - Review the results achieved so far with the customer and plan the next iteration around the spiral.
  - Progressively more complete version of the software gets built with each iteration around the spiral.

Circumstances to use spiral model

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

Comparison of different life-cycle models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model can not be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer

resentment. On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

The following questions have been designed to test the objectives identified for this module:

1. Identify the definite stages through which a software product undergoes during its lifetime.

Ans.: - The definite stages through which a software product undergoes during its lifetime are as follows:

- Feasibility Study
- Requirements Analysis and Specification
- Design
- Coding and Unit Testing
- Integration and System Testing, and
- Maintenance

2. Explain the problems that might be faced by an organization if it does not follow any software life cycle model.

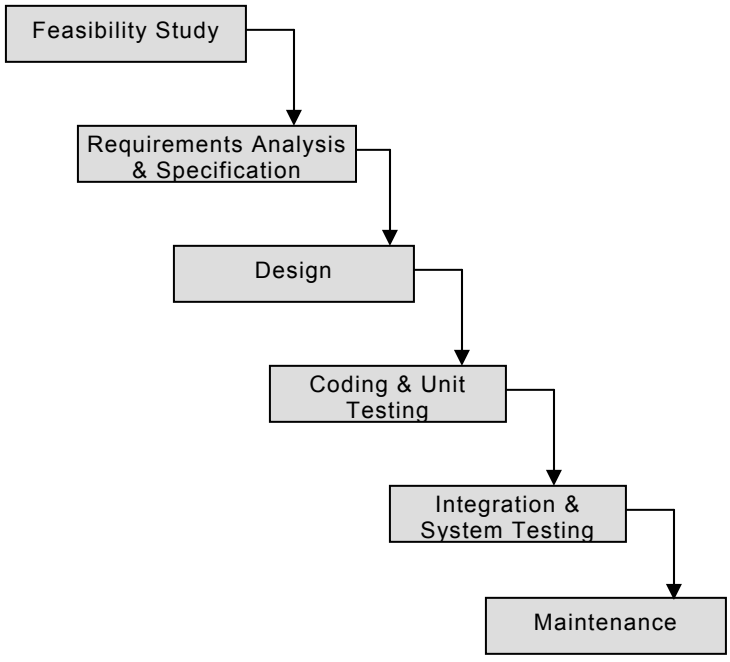
Ans.: - The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.

A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models (such as classical waterfall model, iterative waterfall model, prototyping model, evolutionary model, spiral model etc.) it becomes difficult for software project managers to monitor the progress of the project.

3. Identify six different phases of a classical waterfall model.

Ans.: - The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it can not be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.

Classical waterfall model divides the life cycle into the following phases as shown in fig. 2.1(Classical Waterfall Model):



- Feasibility Study
- Requirements Analysis and Specification
- Design
- Coding and Unit Testing
- Integration and System Testing, and
- Maintenance

4. Identify two basic roles of a system analyst.

Ans.:- For performing requirements analysis activity system analyst collects all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore a system analyst identifies all ambiguities and contradictions in the requirements and resolves them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the system analyst starts requirements specification activity. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

5. Differentiate between structured analysis and structured design.

Ans.: - Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.

6. Identify at least three activities undertaken in an object-oriented software design approach.

Ans.: - In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

7. State why it is a good idea to test a module in isolation from other modules.

Ans.: - During unit testing, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage. So it is always a good idea to test a module in isolation from other modules.

8. Identify why different modules making up a software product are almost never integrated in one shot.

Ans.: - Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out.

9. Mention at least two reasons as to why classical waterfall model can be considered impractical and cannot be used in real projects.

Ans.: - The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

10. Explain what is a software prototype.

Ans.: - A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

11. Identify three reasons for the necessity of developing a prototype during software development.

Ans.: - There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- how screens might look like
- how the user interface would behave
- how the system would produce outputs

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

12. Identify when does a prototype need to develop.

Ans.: - A prototype can be developed when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

13. Identify at least two activities carried out during each phase of a spiral model.

Ans.: - The Spiral model of software development is shown in fig. 2.2. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. 2.2. The following activities are carried out during each phase of a spiral model.

- **First quadrant (Objective Setting)**
  - During the first quadrant, it is needed to identify the objectives of the phase.
  - Examine the risks associated with these objectives.
- **Second Quadrant (Risk Assessment and Reduction)**
  - A detailed analysis is carried out for each identified project risk.
  - Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.
- **Third Quadrant (Development and Validation)**
  - Develop and validate the next level of the product after resolving the identified risks.
- **Fourth Quadrant (Review and Planning)**
  - Review the results achieved so far with the customer and plan the next iteration around the spiral.
  - Progressively more complete version of the software gets built with each iteration around the spiral.

14. Write down the two advantages of using spiral model.

Ans.: - The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

For the following, mark all options which are true.

1. In a classical waterfall model, which phase precedes the design phase ?
□ Coding and unit testing
□ Maintenance
□ Requirements analysis and specification √
□ Feasibility study

2. Among development phases of software life cycle, which phase typically consumes the maximum effort?
□ Requirements analysis and specification
□ Design
□ Coding
□ Testing √

3. Among all the phases of software life cycle, which phase consumes the maximum effort?
□ Design
□ Maintenance √
□ Testing
□ Coding

4. In the classical waterfall model during which phase is the Software Requirement Specification (SRS) document produced?
□ Design
□ Maintenance
□ Requirements analysis and specification √
□ Coding

5. Which phase is the last development phase of a classical waterfall software life cycle?
□ Design
□ Maintenance
□ Testing √
□ Coding

6. Which development phase in classical waterfall life cycle immediately follows coding phase?
□ Design
□ Maintenance
□ Testing √
□ Requirement analysis and specification

7. Out of the following life cycle models which one can be considered as the most general model, and the others as specialization of it?
□ Classical Waterfall Model √
□ Iterative Waterfall Model
□ Prototyping Model
□ Spiral Model

Mark the following as either True or False. Justify your answer.

1. Evolutionary life cycle model is ideally suited for development of very small software products typically requiring a few months of development effort.

Ans.: - False.

Explanation: - The Evolutionary model is very useful for very large problems where it becomes easier to find modules for incremental implementation.

2. Prototyping life cycle model is the most suitable one for undertaking a software development project susceptible to schedule slippage.

Ans.: - False.

Explanation: - The prototype model is suitable for projects whose user requirements or the underlying technical aspects are not well understood.

3. Spiral life cycle model is not suitable for products that are vulnerable to large number of risks.

Ans.: - False.

Explanation: - The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.