

HANDOUT # 1

BASIC SOFTWARE LIFECYCLE

LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out. For example, the design phase might consist of the structured analysis activity followed by the structured design activity.

WHY NEED A SOFTWARE LIFE CYCLE MODEL?

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.

Different software life cycle models

Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model

Different phases of the classical waterfall model

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:

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

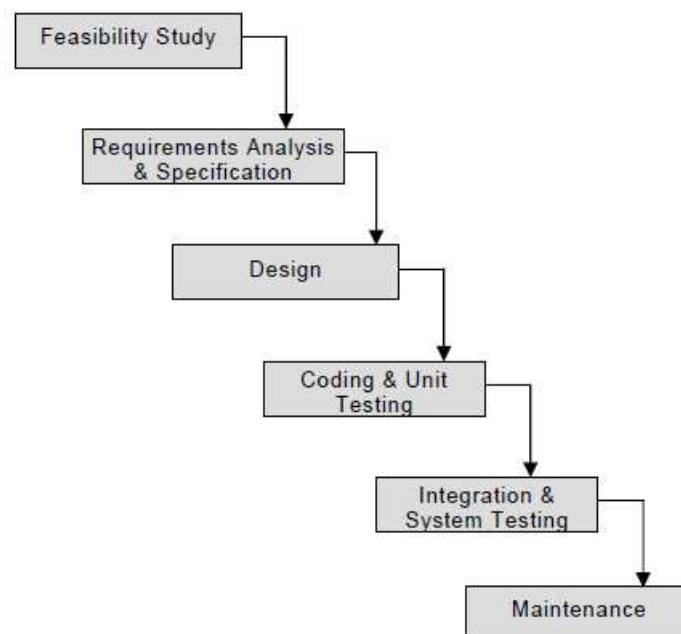


Fig 1: Classical Waterfall Model

Note:

An argument for the waterfall model is that it places emphasis on documentation as well as source code. In less designed & documented methodologies, should team member leave, much knowledge is lost & may be difficult for a project to recover from. Should a fully working design document be present new team members or even entirely new teams should be able to familiarize themselves by reading the documents. It is simple & arguably more disciplined and a structured approach. The

model itself progresses linearly through discrete easily understandable & explainable phases & thus is easy to understand. It also provides easily mark able milestones in the development process. It is perhaps for this reason that the waterfall model is used as a beginning example of a development model in many software engineering courses. The waterfall model also requires that implementers follow the well made, complete design ensuring that the integration of the system proceeds smoothly.

ACTIVITIES IN EACH PHASE OF THE LIFE CYCLE

Activities undertaken during feasibility study: -

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behaviour of the system.
- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.
- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

Activities undertaken during requirements analysis and specification:

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis, and
- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting 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 it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a **Software Requirements Specification (SRS)** document.

The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the non-functional requirements, and the goals of implementation.

Activities undertaken during design:

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

Traditional design approach:

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.

Object-oriented design approach:

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.

Activities undertaken during coding and unit testing:

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

During this phase, 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.

Activities undertaken during integration and system testing:

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. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – Testing: It is the system testing performed by the development team.
- β – Testing: It is the system testing performed by a friendly set of customers.
- Acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

Activities undertaken during maintenance: -

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

SHORTCOMINGS OF THE CLASSICAL WATERFALL MODEL:

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.

PHASE-ENTRY AND PHASE-EXIT CRITERIA OF EACH PHASE:

Phase	Entry Criteria	Exit Criteria
Feasibility study	Project managers or team leaders try to understand what the actual problem is by visiting the client side	Project Managers pick the best solution and determine whether the solution is feasible financially and technically.
Requirements analysis and specification	The required data is collected. After that requirement specification is carried out.	SRS document is produced.
Design	Context diagram and different levels of DFDs are produced according to the SRS document	Module structure (structure chart) is produced.
Coding	Each module (independently compilation unit) of the design is coded. Then each module is tested independently as a stand-alone unit and debugged separately. After this each module is documented individually	The end product of the implementation phase is a set of program modules that have been tested individually but not tested together.
Implementation	Different modules which have been tested individually are integrated in a planned manner	After all the modules have been successfully integrated and tested, system testing is carried out

Software maintenance denotes any changes made to a software product after it has been delivered to the customer. Maintenance is inevitable for almost any kind of product. However, most products need maintenance due to the wear and tear caused by use.