SOFTWARE DEVELOPMENT LIFE CYCLE

Life Cycle Model

- A Software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle.
- It represents all the activities required to make a software product transit through its life cycle phases.
- It captures the order in which these activities are to be undertaken.
- It 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.
- 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.

Need for A Life Cycle Model

- To develop of a software product in a systematic and disciplined manner.
- To make a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and 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.
- Without software life cycle model the entry and exit criteria for a phase cannot be recognized.
- Without software life cycle models it becomes very difficult for software project managers to monitor the progress of the project.

Different Life Cycle Models

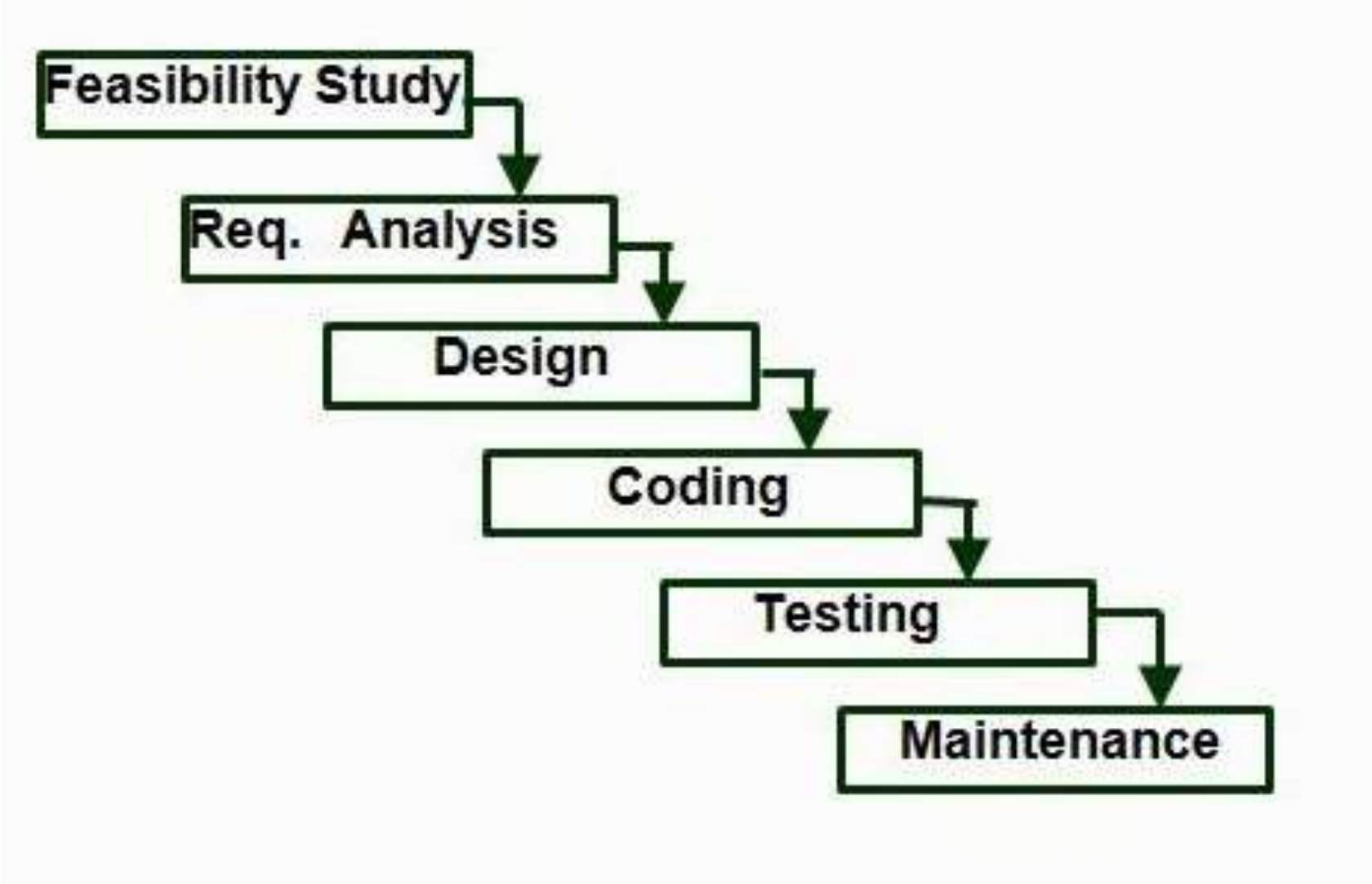
- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model

Classical Waterfall Model

sposto

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 cannot 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



Classical Waterfall Model

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

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 behavior 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.

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
- Requirements specification

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.

Requirements Analysis and Specification

The data collected from a group of users usually contain several contradictions and ambiguities.

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 nonfunctional requirements, and the goals of implementation.

Design

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
- 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 furtherrefined to obtain the detailed design.

Coding and Unit Testing

The purpose of the coding 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.

Integration and System Testing

milon

Integration of different modules is undertaken once they have been coded and unit tested.

All the modules are integrated in a planned manner.

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.

Integration and System Testing

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.

Maintenance

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself.

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.