Software Engineering

Prepared by : Neha Tripathi

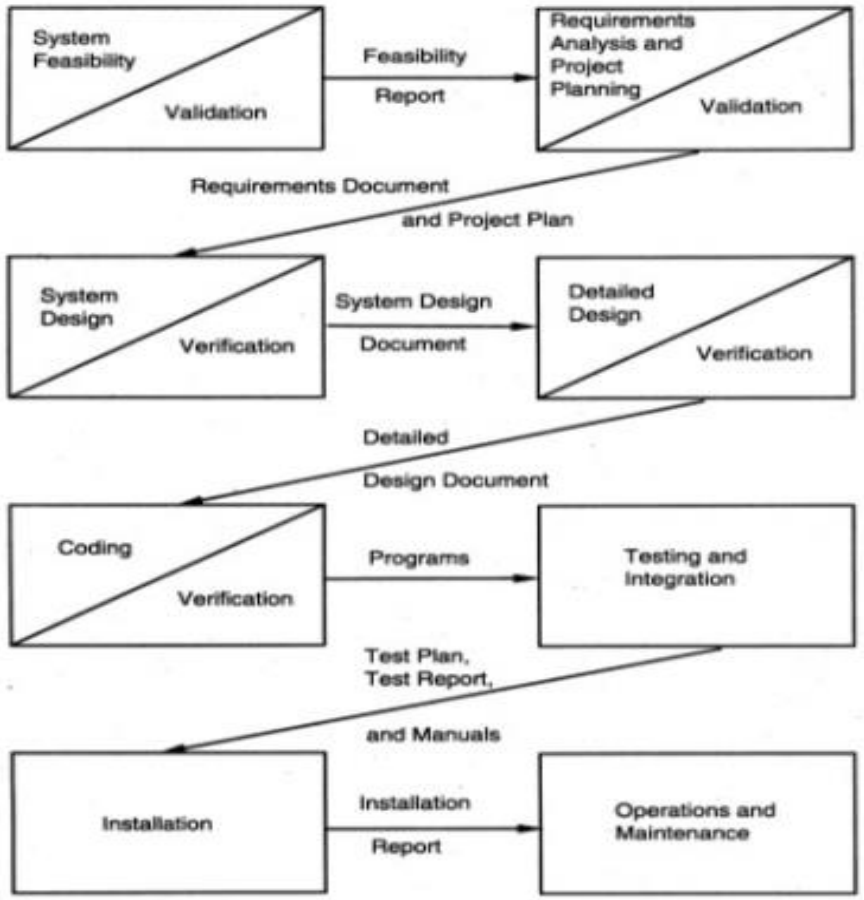
SDLC-Waterfall Model

- **Classical Waterfall Model**
- **Iterative Waterfall Model**

Classical Waterfall Model

- The Waterfall Model was the **first Process Model** to be introduced.
- It is also referred to as a **linear-sequential life cycle model**.
- It is very **simple to understand and use**.
- Classical waterfall model divides the **life cycle into a set of phases** and each phase consists of a series of tasks and has different objectives.
- In a waterfall model, each phase must be completed before the next phase can begin and there **is no overlapping in the phases**.
- **Since the phases fall from a higher level to lower level, like a cascade of waterfall, It's named as the waterfall model.**
- Phases between feasibility study and testing is known as **development phases**.
- Among all life cycle phases **maintenance phase consumes maximum effort.**
- Among development phases, **testing phase consumes the maximum effort**.

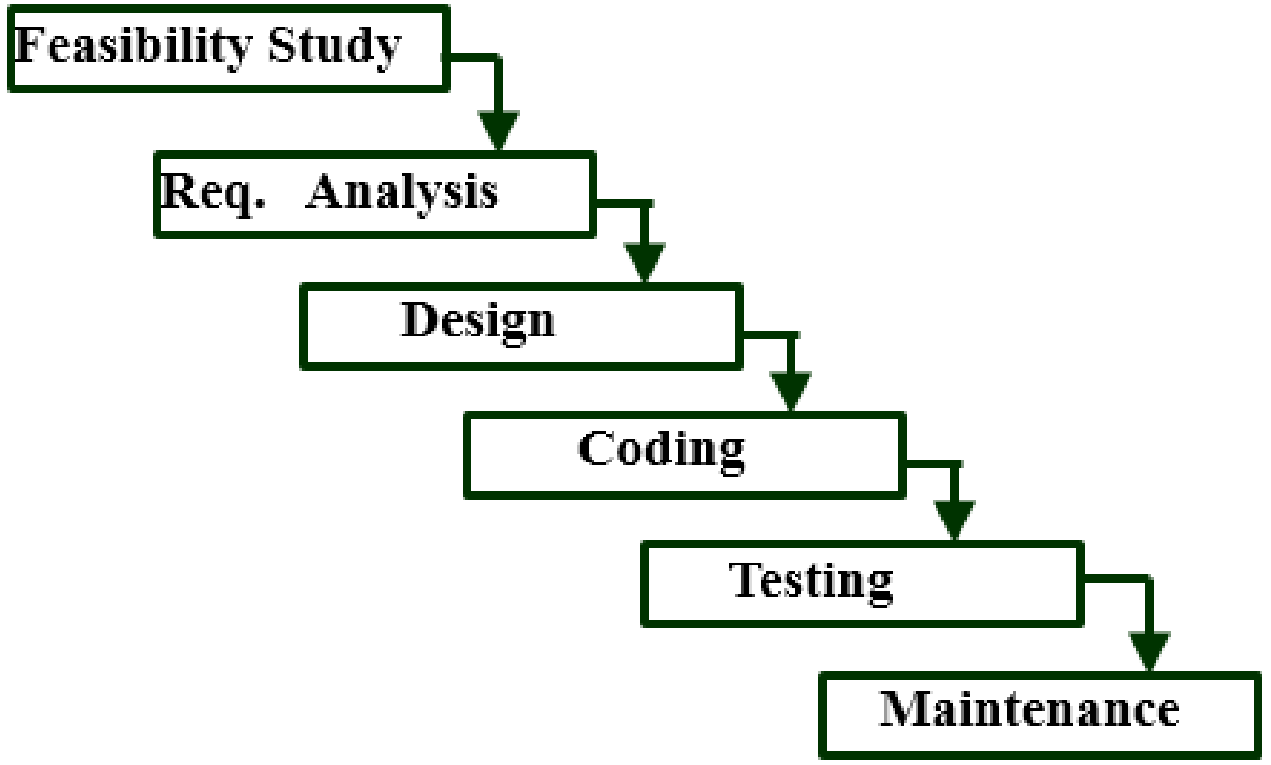
Classical Waterfall Model – representation1



Deliverables of Waterfall Model

- Requirements document (SRS : Software Requirement Specifications)
- Project plan
- System design document
- Detailed design document
- Test plans and test reports
- Source code
- Software manuals (user manual, installation manual)
- Review reports

Classical Waterfall Model – representation2



Phases of Classical waterfall Model

1. Feasibility Study: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. Requirements analysis and specification: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

- **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness and inconsistencies.
- **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design: The aim 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.

4. Coding and Unit testing: In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. Integration and System testing: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists three different kinds of testing activities as described below :

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

6. Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

7. Maintenance: Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance:

- **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

When To Use Waterfall Model-Application

SDLC Waterfall model is used when

- Requirements are initially well known
- Requirements are stable and not changed frequently.
- An application is small.
- There is no requirement which is not understood or not very clear.
- The environment is stable
- The tools and techniques used is stable and is not dynamic
- Resources are well trained and are available.

Waterfall Model - Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

Classical waterfall model suffers from various shortcomings, basically we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

- **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase the efficiency and reduce the cost, phases may overlap.

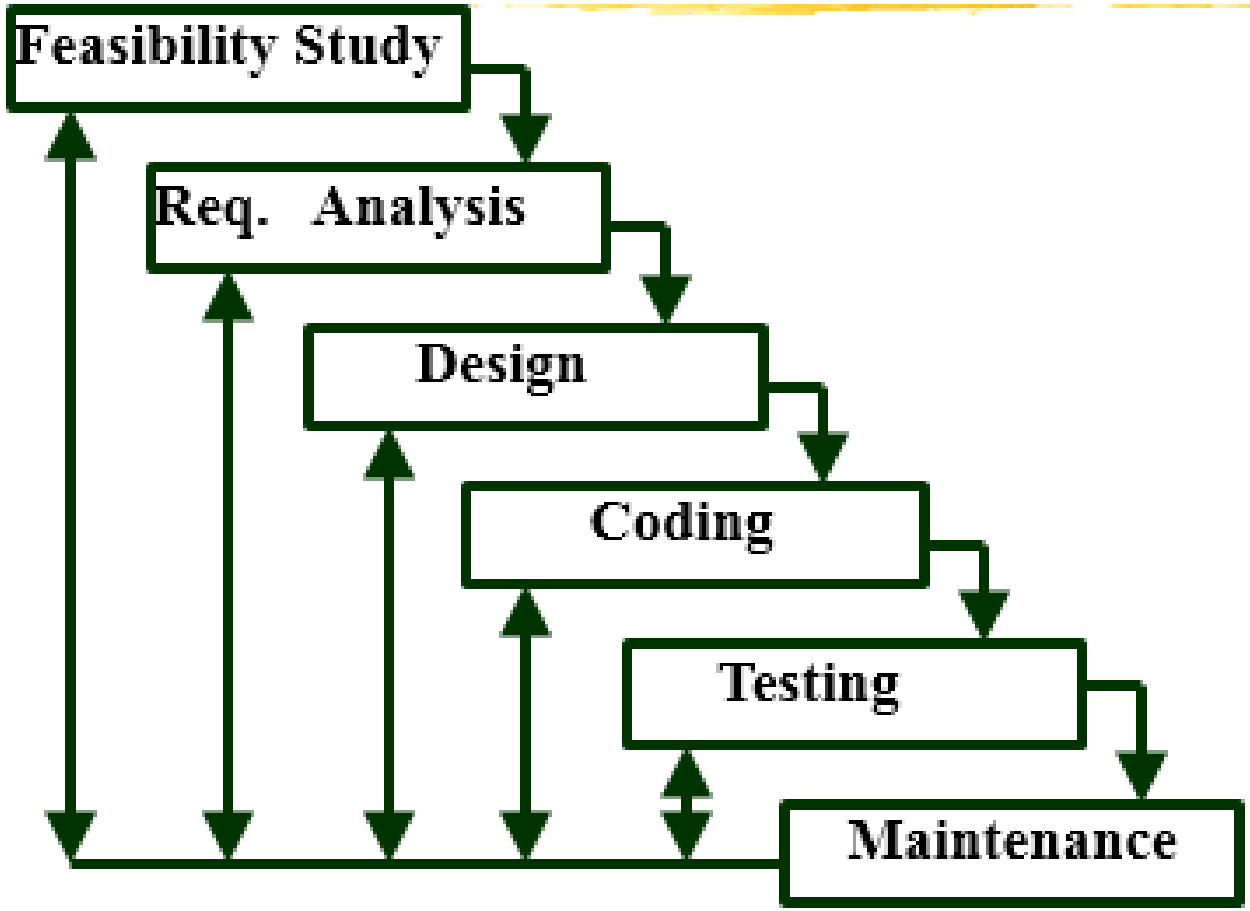
Other drawbacks includes:

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Waterfall Model

- **Classical waterfall model is idealistic:**
  - **assumes that no defect is introduced during any development activity.**
  - **in practice:**
    - **defects do get introduced in almost every phase of the life cycle.**
- **Defects usually get detected much later in the life cycle:**
  - For example, a design defect might go unnoticed till the coding or testing phase.
- **Once a defect is detected:**
  - **we need to go back to the phase where it was introduced**
  - **redo some of the work done during that and all subsequent phases.**
- **Therefore we need feedback paths in the classical waterfall model.**

Iterative Waterfall Model-representation



- Errors should be detected in the same phase in which they are introduced.
- For example:
  - if a design problem is detected in the design phase itself,
    - the problem can be taken care of much more easily
    - than say if it is identified at the end of the integration and system testing phase.
- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle. of detecting errors as close to its point of introduction as possible is known as **phase containment of errors**
- Iterative waterfall model is by far the **most widely used model**.
  - Almost every other model is derived from the waterfall model.

Advantages of Iterative Waterfall Model

- **Feedback Path:** In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases.
- **Simple:** Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.

Drawbacks of Iterative Waterfall Model

- **Difficult to incorporate change requests:** The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting of the development phase. Customer may change requirements after some time but the iterative waterfall model does not leave any scope to incorporate change requests that are made after development phase starts.
- **Incremental delivery not supported:** In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait long for getting the software.
- **Overlapping of phases not supported:** Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.
- **Risk handling not supported:** Projects may suffer from various types of risks. But, Iterative waterfall model has no mechanism for risk handling.
- **Limited customer interactions:** Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

Thank You!