What is a Software?
A software is a collection of instructions, computer programs and associated documentations along with operating procedures that are used to run machines and carry out particular tasks.

What are characteristics of a Software?

- Software is developed or engineered and not manufactured in the classical sense. Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
- Software does not wear out like hardware. As time passes, the failure rate rises as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.
- Although the industry is moving toward component-based assembly, most software Continues to be custom built. Software component should be designed and implemented so that it can be reused in many different programs.

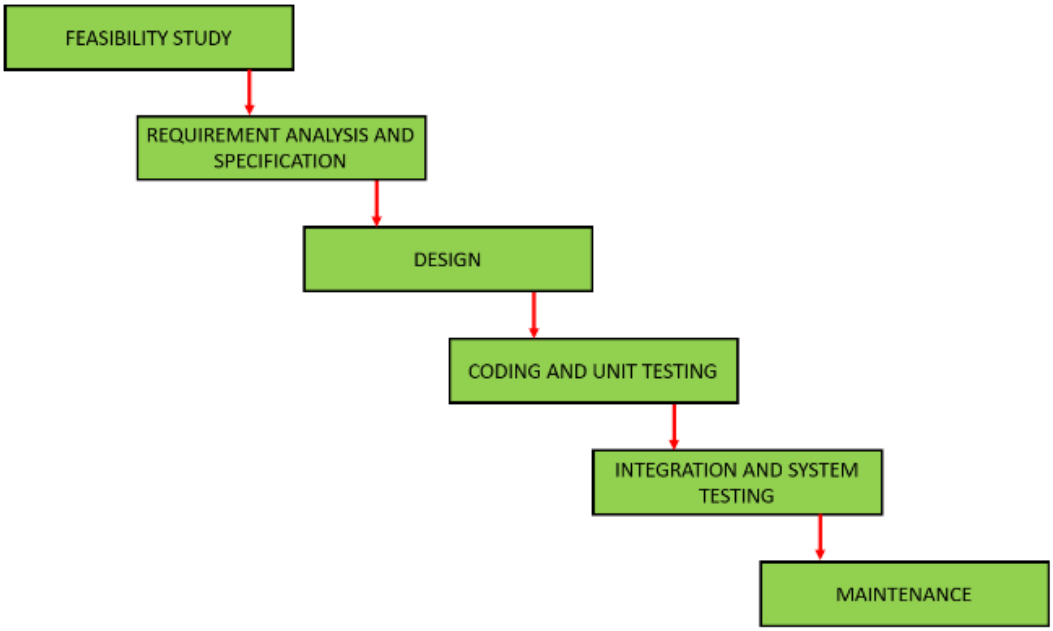
What challenges does a software possess?

- Ensuring the quality of the software to be produced.
- Meeting the growing demand and still be able to maintain the budget.
- To be able to upgrade an "aging software plant"
- Avoiding disastrous time delays.
- Lack of guidance from management.
- Difficulty estimating time and resources.
- Facing lack of resources for the software development
- To control the complexity of software projects.
- Testing, Debugging and maintaining the competitive edge.

What is Software Engineering?
Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of a software.

The Waterfall Model:

- The model is named waterfall because its diagrammatic representation resembles a cascade of waterfall.
- The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modelling, construction, and deployment of the completed software.



1. Feasibility Study: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software by understanding the problem and then determining the various possible strategies to solve the problem.

2. Requirements Analysis and Specification: to understand the exact requirements of the customer and document them properly all the requirements regarding the software are gathered from the customer and then analyzed. These analyzed requirements are documented in a software requirement specification (SRS) document.

3. Design: The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language.

4. Coding and Unit Testing: In the 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 is undertaken soon after they have been coded and unit 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.

6. Maintenance: Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software.

Advantages of the Classical Waterfall Model:

- Easy to Understand: Classical Waterfall Model is very simple and easy to understand.
- Individual Processing: Phases in the Classical Waterfall model are processed one at a time.
- Properly Defined: In the classical waterfall model, each stage in the model is clearly defined.
- Clear Milestones: Classical Waterfall model has very clear and well-understood milestones.
- Properly Documented: Processes, actions, and results are very well documented.
- Reinforces Good Habits: Classical Waterfall Model reinforces good habits like define-before-design and design-before-code.
- Working: Classical Waterfall Model works well for smaller projects and projects where requirements are well understood.

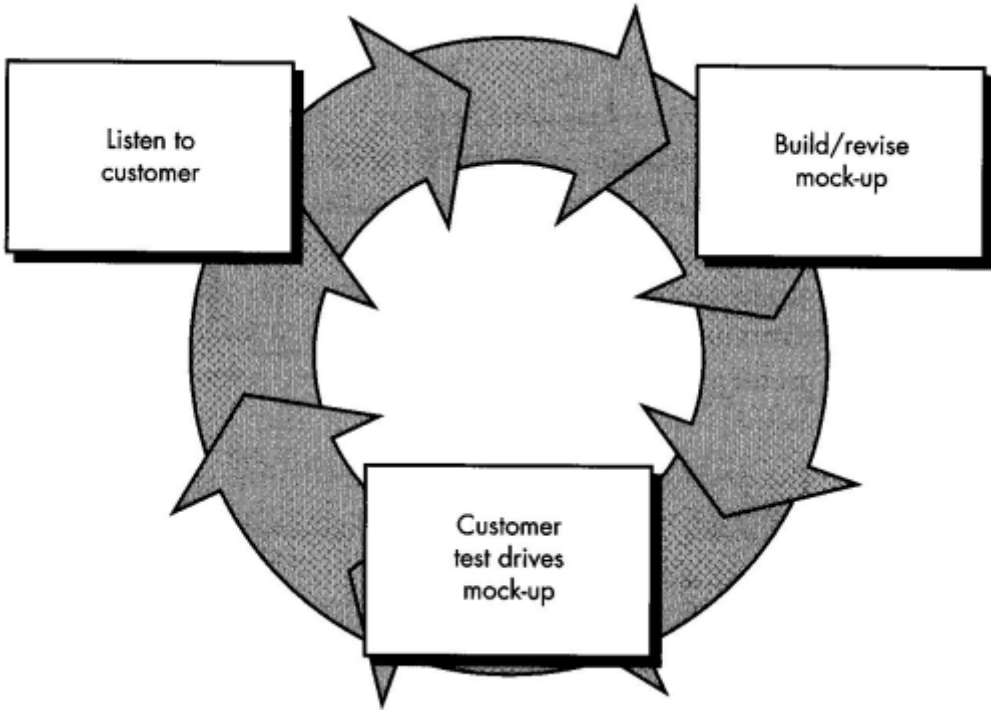
Drawbacks of waterfall model:

- It is difficult to follow the sequential flow in software development process. If some changes are made at some phases then it may cause some confusion.
- The requirement analysis is done initially and sometimes it is not possible to state all the requirements explicitly in the beginning. This causes difficulty in the project.
- The customer can see the working model of the project only at the end. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.
- Linear nature of waterfall model induces blocking states, because certain tasks may be dependent on some previous tasks. Hence it is necessary to accomplish all the dependant tasks first. It may cause long waiting time.

The Prototyping Model:

- This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved.

- The prototyping paradigm begins with communication between the developer and customer where they meet and define the overall objectives for the software and identify whatever requirements are known.
- A quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. Input approaches and output formats). The quick design leads to the construction of a prototype.
- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- Ideally, the prototype serves as a mechanism for identifying software requirements.



Advantages:

- Requirements can be set earlier and more reliably.
- Customer sees results very quickly.
- Customer is educated in what is possible helping to refine requirements.
- Requirements can be communicated more clearly and completely.
- Between developers and clients Requirements and design options can be investigated quickly and cheaply.

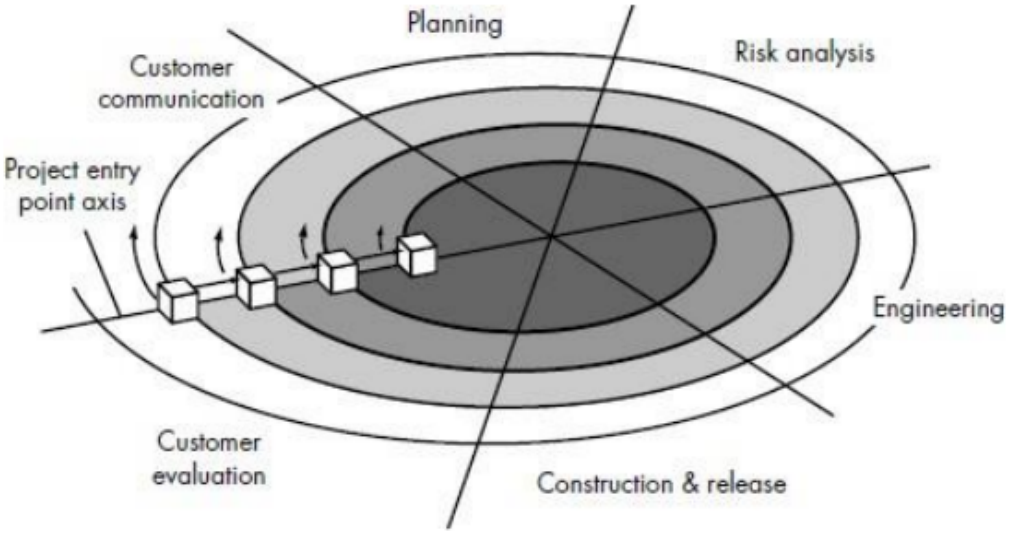
Drawbacks of prototyping:

- In the first version itself, customer often wants “few fixes” rather than rebuilding of the system whereas rebuilding of new system maintains

high level of quality.
- The first version may have some compromises.
- Sometimes developer may make implementation compromises to get prototype working quickly.
- Later on developer may become comfortable with compromises and forget why they are inappropriate.

The Spiral Model:

- The spiral model is an evolutionary software process model that couples the iterative nature of prototyping along with the controlled and systematic aspects of the linear sequential model.
- The spiral model is used by software engineers and is favored for large, expensive and complicated projects.
- The Spiral Model is a risk-driven model, meaning that the focus is on managing risk through multiple iterations of the software development process.
- A key aspect of the process is to make sure that the needs of stakeholders and end-users are taken into account at each phase so that the final product meets their expectations.
- It has two main distinguishing features. One is a cyclic approach for incrementally growing of a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- Using the spiral model, software is developed in a series of incremental releases.
- A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions.



- Customer communication—tasks required to establish effective communication between developer and customer.
- Planning—tasks required to define resources, timelines, and other project related information.
- Risk analysis—tasks required to assess both technical and management risks.
- Engineering—tasks required to build one or more representations of the application.
- Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training). Figure 1.10: Spiral model
- Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage

Spiral Model Advantages:

- Requirement changes can be made at every stage.
- Risks can be identified and rectified before they get problematic.
- It is recommended to use the Spiral Model in large and complex projects.
- The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.

Spiral Model disadvantages:

- The Spiral Model is much more complex than other SDLC models.
- Spiral Model is not suitable for small projects as it is expensive.
- It is based on customer communication. If the communication is not proper then the software product that gets developed will not be up to the mark.
- It demands considerable risk assessment. If the risk assessment is done properly then only the successful product can be obtained.
- The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.

What is Agility?

- Agility is the ability to respond quickly to the changing needs.
- It encourages team structures and attitudes that make effective communication among all stakeholders.
- It emphasizes rapid delivery of operational software and de-emphasizes the importance of intermediate work products.
- It adopts the customer as a part of the development team.
- It helps in organizing a team that it is in control of the work performed.

Agility and the Cost of Change:

- The cost of change in software development increases non linearly as a project progresses.
- It is relatively easy to accommodate a change when software team gathered its requirements.
- The costs of doing this work are minimal, and the time required will not affect the outcome of the project.
- Cost varies quickly, and the cost and time required to ensure that the change is made without any side effects is nontrivial.
- An agile process reduces the cost of change because software is released in increments and changes can be better controlled within an increment.
- Agile process "flattens" the cost of change curve (Figure 1.11, shaded, solid curve), allowing a software team to accommodate changes late in a software project without dramatic cost and time impact.
- When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.

What is an Agile process:

An Agile Process is characterized in a manner that addresses a number of key assumptions about the majority of software project:

1. It is difficult to predict which software requirements will persist and which will change.
2. It is difficult to predict how customer priorities will change.
3. It is difficult to predict how much design is necessary before construction.
4. Analysis, design, construction, and testing are not as predictable.

What are the Agility principles?

1. To satisfy the customer through early and continuous delivery of software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months.
4. 'Customers and developers must work together daily throughout the project.
5. Build projects around motivated individuals.
6. Emphasis on face-to-face communication.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. Self-organizing teams produce the best architectures/requirements/design.
12. The team reflects on how to become more effective at regular intervals.

What are the human factors in Agile Development?
Competence - Software related skills, Knowledge of process.
Common Focus-deliver a working software Increment to customer within time promised.
Collaboration-Among team members and other stakeholders
Decision making ability-Team is given Autonomy
Fuzzy Problem Solving ability-Today problem may not be tomorrow (but Lesson learn may be benefit to the team later)
Mutual Trust And Respect

Extreme Programming: XP is built upon values, principles, and practices, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements.
Extreme Programming (XP) takes an 'extreme' approach to iterative development.
New versions may be built several times per day;
Increments are delivered to customers every 2 weeks;
All tests must be run for every build and the build is only accepted if tests run successfully.
Extreme programming uses an object-oriented approach for software development.
There are four framework activities involved in XP Process are shown in
1. Planning
2. Designing
3. Coding
4. Testing

- Planning: Define the scope of the project and prioritize tasks. Create user stories and estimate the effort required for each.
- Design: Develop a high-level architectural design and make initial design decisions. Keep the design simple and adaptable.
- Coding: Write code in pairs (pair programming) or individually, focusing on test-driven development (TDD). Write unit tests before writing the actual code.
- Testing: Continuously test the software to ensure it meets the specified requirements. Automated unit tests and frequent integration testing are essential.