20AI501 SOFTWARE ENGINEERING AND TESTING METHODOLOGIES

Unit1: SOFTWARE PROCESS MODELS

- ✓ Introduction to Software Engineering
- ✓ Software Development Life Cycle(SDLC)
- ✓ Types of Software Process Models
- ✓ Prescriptive Process Models

What is Engineering?

Engineering is the **application of scientific and practical knowledge** to invent, design, build, maintain, and improve frameworks, processes, etc.

What is software?

- Software is a **collection of instructions and data** that tell a computer how to work.
- Software comprises the entire set of programs, procedures, and routines associated with the operation of a computer system.
- A set of instructions that directs a computer's hardware to perform a task is called a program, or software program.

Various categories of software:

- System software
- Application software
- Engineering and scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software…

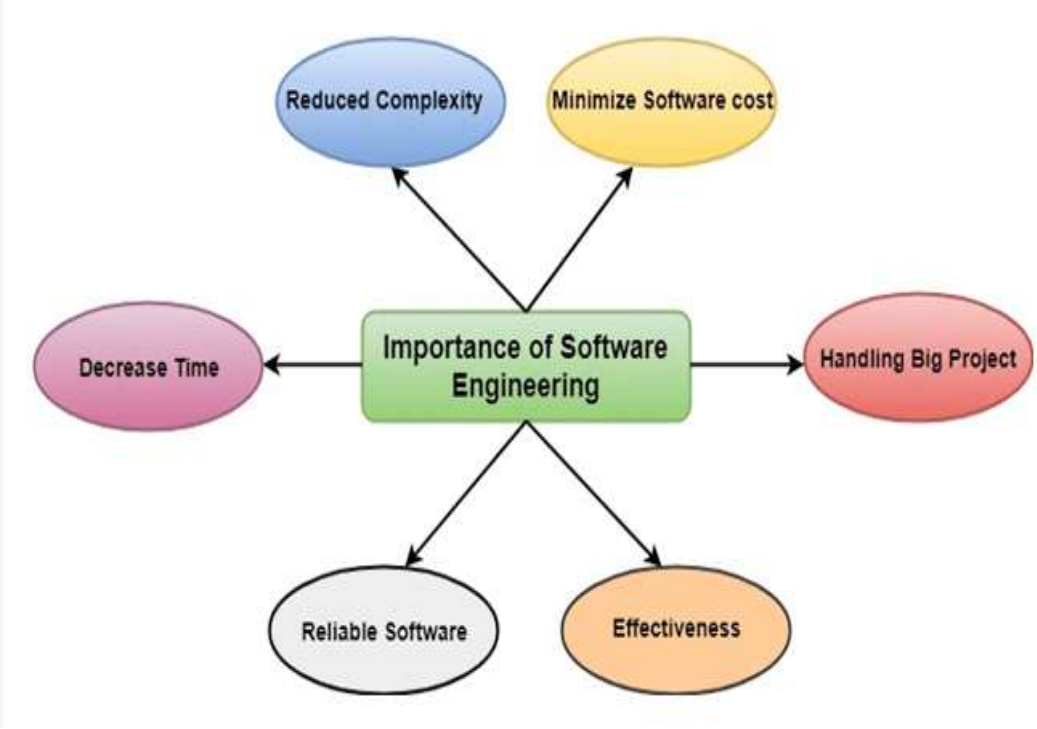
What is Software Engineering?

- S**oftware Engineering** is a **systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system**.
- Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Characteristics of software

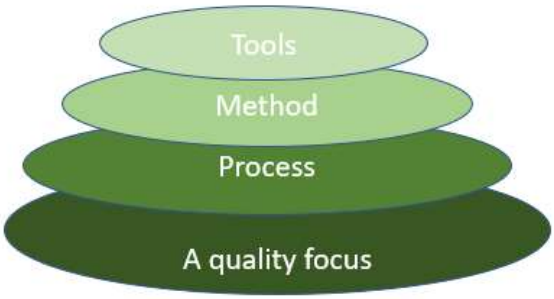
- Software is **developed or engineered**, but it is not manufactured in the classical sense.
- Software **does not wear out**, but it deteriorates due to change.
- Software is **custom built** rather than assembling existing components.

Importance of Software Engineering



Software Engineering-A Layered Technology

- All the **layers are connected** and each layer **demands the fulfillment of the previous** layer.



Quality focus:

- Defines the continuous **process improvement** principles of software.
- Provides **integrity** that means providing **security** to the software.
- Focuses on **maintainability and usability**.

Process:

- **Base layer** of software engineering.
- Process defines a **framework/activiites/task that must be established for the effective delivery of software** engineering technology.
- **Binds all the layers together** which enables the development of **software before** the **deadline** or on time.
- Process activities are **Communication, Planning, Modeling, Construction, and Deployment.**

Methods:

- Answers to all "**how-to-do**" questions are given by method.
- It has the **information of all the tasks** which includes communication, requirement analysis, design modeling, program construction, testing, and support.

Tools:

- Software engineering tools provide a **self-operating system for processes and methods**.
- Tools are integrated which means information created by one tool can be used by another.

Software process:

- A **process** is a **collection of activities, actions and tasks** that are performed when some **work product is to be created**.
- A **software process** is a set of **related activities that leads to the production of a software** system.

Four fundamental activities in software processes:

1. **Software specification:** The **functionality of the** software **and constraints** on its operation must be **defined**.
2. **Software development:** The **software** to meet the **specification** must be **produced**.
3. **Software validation:** The **software** must be **validated** to ensure that it does **what** the **customer wants**.
4. **Software evolution:** The **software** must **evolve** to meet **changing customer needs**.

A Process Framework

- Identifies a **number of framework activities** applicable to all software projects.
- Software Engineering Process Framework activities are complemented by a number of umbrella activities.
- Umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change and risk.

Generic Process Framework for software Engineering encompasses five activities:

• Communication
• Planning
• Modeling
• Construction
• Deployment

Typical Umbrella activities include:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

Software Development Life Cycle:

- The Software Development Life Cycle (SDLC) is **a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time**.
- The goal of the SDLC is to produce **superior software that meets and exceeds all customer expectations and demands**.

1. Requirement and Planning

- The requirement is the **first** stage in the SDLC process.
- It is conducted by the **team members** with **inputs from all the stakeholders and domain experts** in the industry.
- **Planning** for the **quality assurance requirements and recognition of the risks** involved is also done at this stage.
- This stage gives a **clearer picture of the scope of the entire project** and the anticipated issues, opportunities, and directives which triggered the project.
- Requirements Gathering stage need teams to **get detailed and precise requirements**. This helps companies to finalize the necessary timeline to finish the work of that system.

2. Analysis

- Project **goals are converted into the defined system functions** that the organization intends to develop.

Primary activities:

1. Gathering business requirement
2. Creating process diagrams
3. Performing a detailed analysis

Feasibility study

- Once the requirement analysis phase is completed the next step is to **define and document software needs.**
- This process conducted with the help of '**Software Requirement Specification'(SRS)** document.
- It includes **everything** which should be **designed and developed during the project** life cycle.

3. Design

- The system and software design documents are prepared as per the requirement specification document. This helps **define overall system architecture**.
- This phase includes **business rules, pseudo-code, screen layouts, and other necessary documentation.**
- Primary activities:
    - Designing the IT infrastructure
    - Designing the system model

4. Coding/Implementation:

- In this phase, **developers start build the entire system** by writing code using the chosen programming language.
- In the coding phase, tasks are divided into units or modules and assigned to the various developers.
- It is the longest phase of the Software Development Life Cycle process.

5. Testing and Integration

Testing

- Once the software is complete, and it is **deployed in the testing environment**.
- The testing team starts **testing the functionality** of the entire system.
- This is done to verify that the entire application works according to the **customer requirement.**
- During this phase, QA and testing team may find some bugs/defects which they communicate to developers.
- The **development team fixes the bug** and send back to QA for a re-test.
- This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Installation/Deployment

- Once the **software testing phase is over and no bugs** or errors left in the system then the final deployment process starts.
- Based on the feedback given by the project manager, **the final software is released and checked for deployment issues if any.**

6. Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- **Bug fixing** – bugs are reported because of some scenarios which are not tested at all
- **Upgrade** – Upgrading the application to the newer versions of the Software
- **Enhancement** – Adding some new features into the existing software

Verification and Validation

Verification

- **Verification** is the **process of checking that a software achieves its goal without any bugs.**
- It is the process to ensure whether the product that is developed is right or not.
- **Static testing**.
- Verification means **Are we building the product right?**

Validation

- **Validation** is the process of checking whether the software product is up to the mark or in other words product has high level requirements.
- Validation is the **Dynamic testing**.
- Validation means **Are we building the right product?**

Software Process Model:

- **Abstract representation of a process** that presents a description of a process from some particular perspective.

Different types of Software Process Model

- Waterfall model.
- V model.
- Incremental model.
- RAD model.
- Agile model.
- Iterative model.
- Spiral model.
- Prototype model.

Prescriptive process model

- **Define a prescribed set of process elements and a predictable process work flow.**

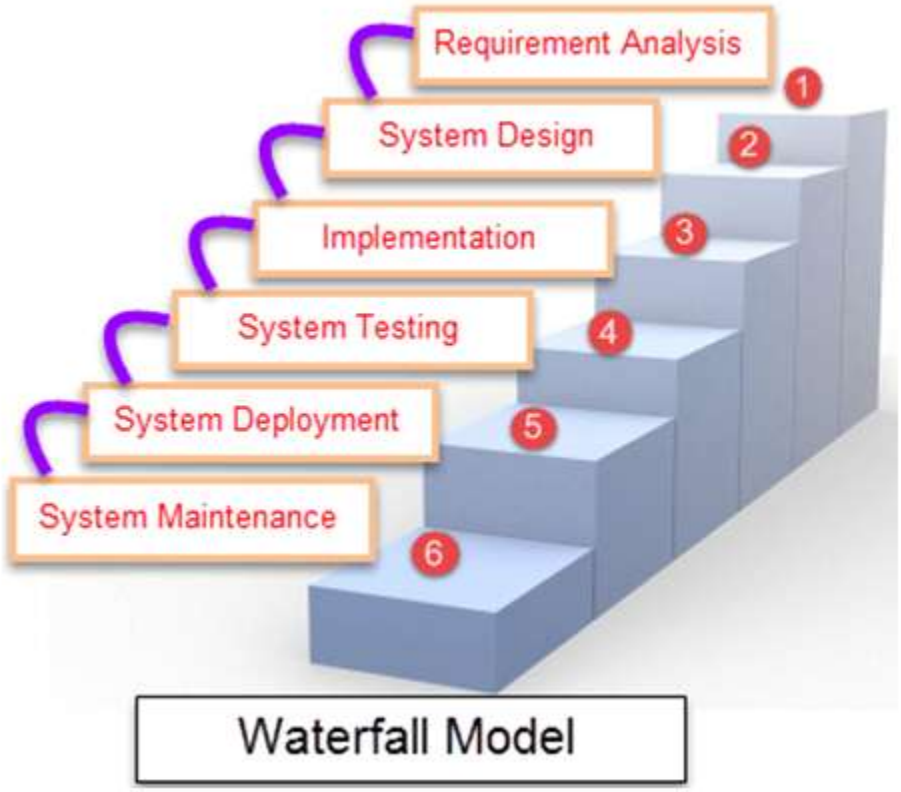
Types:

- Waterfall Model
- Incremental Process Model
- RAD Model

Waterfall Model:

- **Waterfall Model** is a **sequential** model that **divides software** development into **pre-defined phases**.
- **Each phase must be completed before the next phase can begin with no overlap between the phases.**
- It was introduced in 1970 by Winston Royce.

Different Phases of Waterfall Model



1. **Requirement Gathering:** Detailed **requirements** of the software system to be developed are **gathered from client**
2. **Design:** Plan the **programming language**, **database**, other **high-level technical** details of the project
3. **Built**: **Code** the software.

4. **Test**: Test the software to **verify that it is built as per the specifications given by the client**.
5. **Deployment**: **Deploy** the **application** in the respective environment
6. **Maintenance** : Once your system is ready to use, you may **later require change the code as per customer request**

When to use SDLC Waterfall Model?

- Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

Advantages:

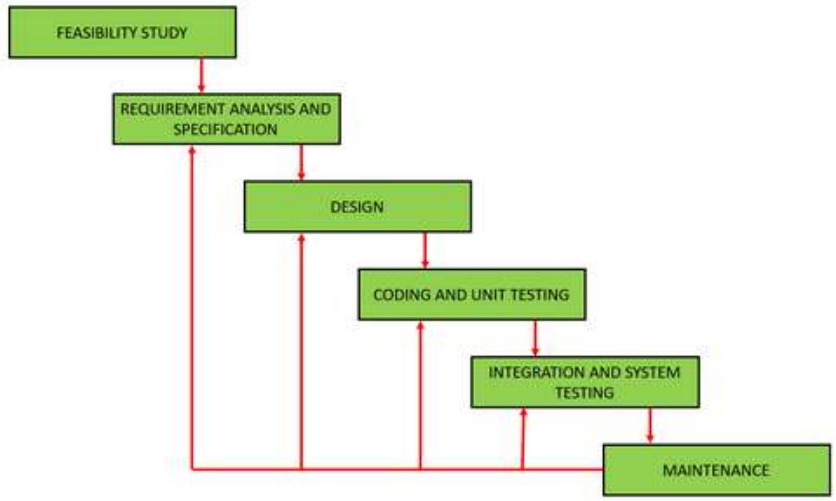
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.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

Disadvantages:

- 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

- The **classical waterfall** model is **hard** to use.
- It is almost the **same as the classical waterfall model except some changes are made to increase the efficiency** of the software development.
- The **iterative waterfall model provides feedback paths from every phase to its preceding phases**, which is the main difference from the classical waterfall model.
- Feedback paths introduced by the iterative waterfall model are shown in the figure below.



- **When errors are detected at some later phase, these feedback paths allow correcting errors committed by programmers during some phase**.
- The **feedback** paths allow the **phase to be reworked in which errors are committed** and these **changes are reflected in the later phases**.
- But, there is **no feedback path to the stage** – feasibility study, because **once a project has been taken, does not give up the project** easily.
- It is **good to detect errors in the same phas**e in which they are committed.
- It **reduces the effort and time** required to **correct the errors**.

Advantages of Iterative Waterfall Model:

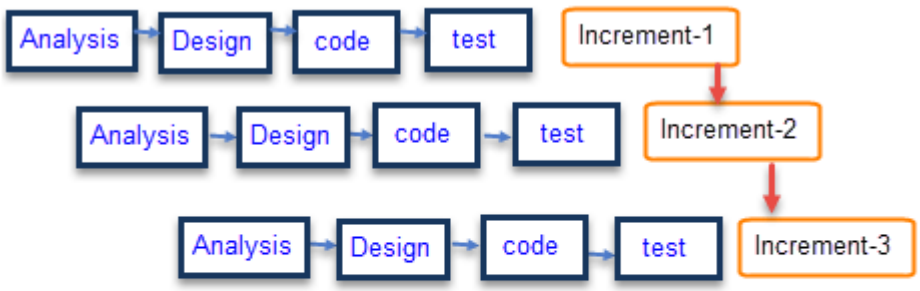
- Feedback Path
- Simple
- Cost-Effective
- Well-organized

Disadvantages of Iterative Waterfall Model:

- Difficult to incorporate change requests
- Incremental delivery not supported
- Overlapping of phases not supported
- Risk handling not supported
- Limited customer interactions

Incremental Model:

- Incremental Model is a process of software development where **requirements are broken down into multiple standalone modules** of software development cycle.
- **Incremental** development is **done** in steps from **analysis design, implementation, testing/verification, maintenance.**
- Each iteration passes through the **requirements, design, coding and testing phases**.
- And each **subsequent release** of the system **adds function** to the previous release until all designed functionality has been implemented.
- The **first increment** is often a **core** product where the **basic requirements** are **addressed**, and **supplementary features are added** in the **next increments**.



Incremental Model

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.
- Projects with new Technology
- Requires good planning and design.

Advantages of Incremental Model

- The software will be generated quickly during the software life cycle
- Throughout the development stages changes can be done
- Errors are easy to be recognized.
- Easier to test and debug
- More flexible and less expensive.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantages of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.