Software Engineering

What is Software -

software is a program along with proper documentation (requirement analysis , design , coding , testing) and user manuals which may includes installation guide and other manuals .

software = program + documentation

Program vs. Software

Feature	Program	Software
Definition	A set of instructions written in a programming language to perform a specific task.	A collection of programs, data, and documentation that work together to provide functionality.
Size & Scope	Usually small and designed for a specific task.	Larger, often consisting of multiple programs working together.
Dependency	May work independently or as part of software.	Consists of multiple programs and other resources.
Examples	A Python script to add two numbers.	Microsoft Office (which includes Word, Excel, PowerPoint, etc.).

Software Engineering

Difference Between System Software and Application Software

Feature	System Software	Application Software
Definition	Software that manages hardware and system operations.	Software designed to perform specific user-oriented tasks.
Purpose	Acts as an interface between hardware and users or applications.	Helps users accomplish specific tasks like editing, gaming, or browsing.
Dependency	Runs before application software and provides an environment for it.	Depends on system software to function.
Examples	Operating System (Windows, Linux, macOS), Drivers, Utility Programs.	Microsoft Word, Google Chrome, Adobe Photoshop.

<u>What is Software Engineering -</u>

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

or

[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.

Software Engineering

Objectives of Software Engineering

1. **Develop High-Quality Software** – Ensure the software is reliable, efficient, and secure.
2. **Enhance Maintainability** – Software should be easy to modify and update over time.
3. **Improve Efficiency** – Optimize resource usage (memory, processing power, etc.).
4. **Ensure Scalability** – Software should support future growth and new features.
5. **Reduce Development Costs** – Minimize unnecessary expenses through effective planning and design.
6. **Ensure Reusability** – Develop modular components that can be reused in other projects.
7. **Meet User Requirements** – Deliver software that fulfills customer needs and expectations.
8. **Minimize Risks** – Identify and mitigate software failures or security vulnerabilities.
9. **Standardization & Documentation** – Follow coding standards and document the software for better collaboration and maintenance.

Key Principles of Software Engineering

1. **Abstraction** – Simplify complex systems by focusing on essential details while hiding unnecessary complexities.
2. **Modularity** – Break software into smaller, reusable, and manageable components or modules.

Software Engineering

3. **Scalability** – Design software that can handle increasing workloads without performance issues.
4. **Maintainability** – Write clean, well-documented, and structured code that is easy to update and fix.
5. **Reusability** – Develop components that can be used in multiple projects, reducing redundancy.
6. **Security** – Implement security measures to protect software from threats and vulnerabilities.
7. **Performance Optimization** – Ensure the software runs efficiently, utilizing minimal resources.
8. **Reliability** – The software should work correctly under different conditions and avoid crashes.
9. **User-Centered Design** – Prioritize user experience (UX) and usability.
10. **Process Orientation** – Follow well-defined development methodologies like Agile, Waterfall, or DevOps.

Advantages -

1. Improved quality
2. Increased Productivity
3. Better maintability
4. Increased Customer Satisfaction
5. High Upfront Cost
6. Complexity
7. Better Security
8. Limited Flexiblity

Software Engineering

Software Development

Software development is the process of designing, coding, testing, deploying, and maintaining software applications. It involves converting user requirements into a working software product through systematic planning and execution.

It includes various stages, such as requirement gathering, system design, implementation, testing, deployment, and maintenance. Software development can be done using different methodologies like Agile, Waterfall, Scrum, and DevOps depending on the project's needs.

Types of Software Development

1. **Web Development** – Building websites and web applications using HTML, CSS, JavaScript, React, Node.js, etc.
2. **Mobile Development** – Creating mobile apps for Android (Kotlin, Java) and iOS (Swift).
3. **Desktop Application Development** – Developing software for computers (C++, Java, Python).
4. **Game Development** – Building video games using engines like Unity or Unreal Engine.
5. **Embedded Systems Development** – Programming microcontrollers and IoT devices.
6. **AI & Machine Learning Development** – Creating intelligent software using TensorFlow, PyTorch, etc.
7. **Cloud Computing Development** – Building scalable applications using AWS, Google Cloud, or Azure.
8. **Cybersecurity Software Development** – Developing security tools like antivirus software and encryption systems.

Software Engineering

SDLC - Software Development Life Cycle

The **Software Development Life Cycle (SDLC)** is a structured approach used by software organizations to develop high-quality software efficiently. It consists of a step-by-step process that helps in planning, designing, building, testing, deploying, and maintaining software applications.

SDLC ensures that the **end product** meets customer expectations while staying within budget and time constraints. It also helps in **reducing risks, improving quality, and enhancing productivity** throughout the software development process.

Stages of SDLC

1. Planning and Requirement Analysis

📌 *Purpose:* Understanding and gathering requirements for the software.

- In this stage, software engineers analyze the needs of the customers or business.
- Inputs come from customers, sales teams, market research, and industry experts.
- Developers **define the project scope, objectives, risks, cost estimation, and feasibility.**
- A **basic project outline** is created, forming the foundation of the entire software.
- The **quality of the software** depends heavily on effective planning.

✅ *Key Output:* Requirement Analysis Document (RAD)

---

2. Defining Requirements

📌 *Purpose:* Finalizing and documenting all software requirements.

Software Engineering

- All software requirements are **documented and approved** by stakeholders, customers, and market analysts.
- A key document called **Software Requirement Specification (SRS)** is created, which includes:
    - Functional requirements (what the software should do)
    - Non-functional requirements (performance, security, scalability)
    - Hardware and software dependencies
- This ensures that **all project stakeholders have a common understanding of the product.**

✅ *Key Output:* Software Requirement Specification (SRS)

---

3. Designing Architecture

📌 *Purpose:* Creating the blueprint for software development.

- Developers and architects use the **SRS document** to design the software architecture.
- They create multiple architectural models and document them in **Design Document Specification (DDS).**
- Market analysts and stakeholders evaluate these models, and the **best feasible design is selected.**
- The chosen design includes:
    - **Data Flow Diagrams (DFD)**
    - **Database Schema**
    - **User Interface (UI) Prototypes**
    - **Technology Stack Selection (Frontend, Backend, Database, Frameworks)**

Software Engineering

✅ *Key Output:* Design Document Specification (DDS)

---

4. Developing the Product

📌 *Purpose:* Writing the actual software code.

- Developers begin coding based on the **approved design** from the DDS.
- The coding follows **industry standards, security protocols, and best practices.**
- Developers use various programming languages like **C++, Java, Python, JavaScript, etc.**
- **Development tools like compilers, debuggers, and interpreters** are used.
- Developers may use methodologies like **Agile, DevOps, or Waterfall** for better collaboration and efficiency.

✅ *Key Output:* Source Code and Software Modules

---

5. Product Testing and Integration

📌 *Purpose:* Ensuring the software is error-free and meets requirements.

- Testing starts with **unit testing** (individual components) and progresses to **integration testing** (combining components).
- Different testing methods are used, such as:
    - **Functional Testing** – Verifies software functions as expected.
    - **Performance Testing** – Tests speed, load, and scalability.
    - **Security Testing** – Identifies vulnerabilities.

    - **User Acceptance Testing (UAT)** – Ensures it meets customer expectations.
- Bugs are **identified, fixed, and retested** before moving forward.

✅ *Key Output:* Tested Software & Bug Reports

---

6. Deployment and Maintenance

📌 *Purpose:* Delivering the software to users and keeping it updated.

- After successful testing, the **final product is deployed in stages** (pilot testing, phased rollout, or full deployment).
- It is **monitored in real-world conditions** to identify any issues.
- If required, updates and patches are released to **fix bugs, improve performance, and add new features.**
- **Maintenance is ongoing**, ensuring the software remains up-to-date and efficient.

✅ *Key Output:* Live Software & Support Services

Additional Aspects in SDLC

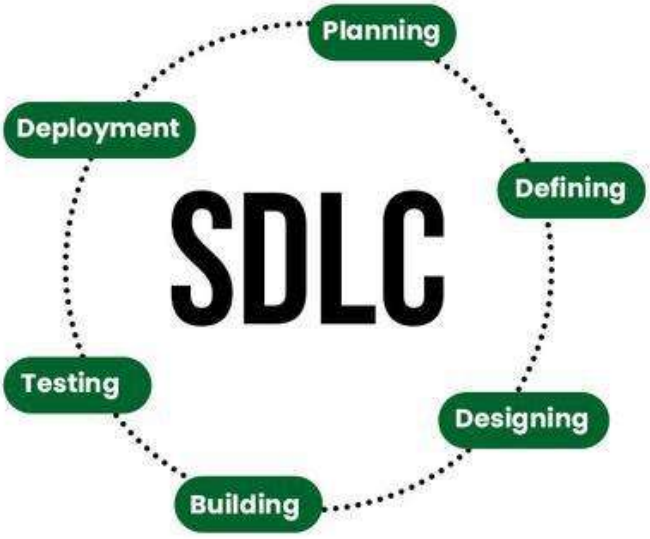
📄 **Documentation**

- Documentation is created for both **developers and end-users**.
- It helps in understanding the software's functionality, troubleshooting, and future upgrades.

🎓 **Training & Support**

- Employees and users are trained to use the software effectively.
- Customer support is provided to assist users and resolve issues.

Software Engineering



Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
Planning & Requirement Analysis	Defining Requirements	Design	Development	Testing	Deployment & Maintenace
Planning	Defining	Design	Development	System Testing	Deployment and Maintenace
Define Project Scope	Functional Requirement	HLD	Coding Standard	Manual Testing	Release Planning
Set Objectives and Goals	Technical Requirement	LLD	Scalable Code	Automated Testing	Deployment Automation
Resource Planning	Requirement Reviews & Approved		Version Control		Maintenance
			Code Review		Feedback

Why is SDLC Important?

- Ensures **structured development** and prevents project failures.
- Reduces **errors and security risks** by following a step-by-step approach.
- Improves **software quality and performance** with thorough testing.
- Helps in **better cost and time estimation** for project completion.
- Makes **maintenance easier** by keeping a clear record of all changes.

Software Engineering

Major Problems in Software Development –

1. **Resistance to change** –
   a. Difficulty in adapting to new technologies or processes
   b. Reluctance to modify established practices or minddets
   c. Rapid technology advancement
2. **Inadequate Requirement Gathering -**
   a. Ambiguous or incomplete requirements
   b. Lack of communication between stakeholders
3. **Poor Project Management -**
   a. Inadequate Planning , monitoring and control
   b. Lack of risk assessment and mitigation
   c. Multiplicity of software development life cycle.
   d. selection of wrong Technologiesor tool for development
4. **Insufficient Time and Budget -**
   a. Unrealistic deadlines and resource constraints
   b. Inefficient resource allocation and prioritization
5. **Lack of Skilled Person**

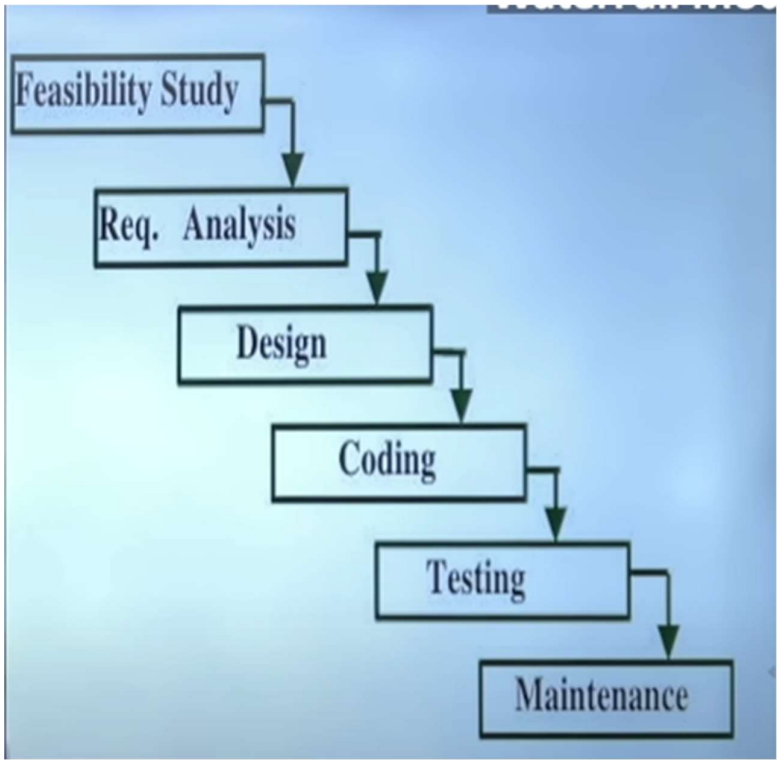
Software Quality Attributes -

- **Correctness –** The ability of the software to perform its intended tasks effectively and meet user requirements.
- **Usability –** simple to use , learn and navigate
- **Reliability** – The Softwares's consistency in producing accurate results and maintaining performance over time.
- **Efficiency** – The optimal use of system resources such as memory and processing power.
- **Maintainability** – ease of updating , modifying and fixing the software to accommodate changing requirements or fix issues
- **Portability** – The ability of the software to operate on different platforms or environments without significant modifications .

Software Engineering

- **Scalability** – software's capacity to handle increased workloads or user demands without compromising performance .
- **Security**
- **Modularity** – The degree to which the software's components are organized into separate , manageable units that can be independentaly developed or updated.
- **Reusablity**
- **Testability**

Waterfall Model –



Key Features -

1. Sequential Process

2. Well Defined Process

3. Strong Documentation

Software Engineering

4. Clear Ptoject Timeline
5. Low Customer Involvement
6. Best for Small & Well – Defined Projects

Advantages of the Waterfall Model

Advantage	Explanation
1. Simple & Easy to Use	**Each phase has a clear start and end, making it easy to manage.**
2. Clear Structure & Process	**Since each phase is well-defined, there is no confusion in project execution.**
3. Strong Documentation	**Helps in understanding project flow, training new team members, and future maintenance.**
4. Well-Suited for Small Projects	**Works best when requirements are clear and do not change.**
5. Easy to Manage & Track Progress	**Since everything is pre-planned, it is easy to track project status and deadlines.**
6. Better for Fixed Budget Projects	**As everything is planned in advance, budget estimation is more accurate.**

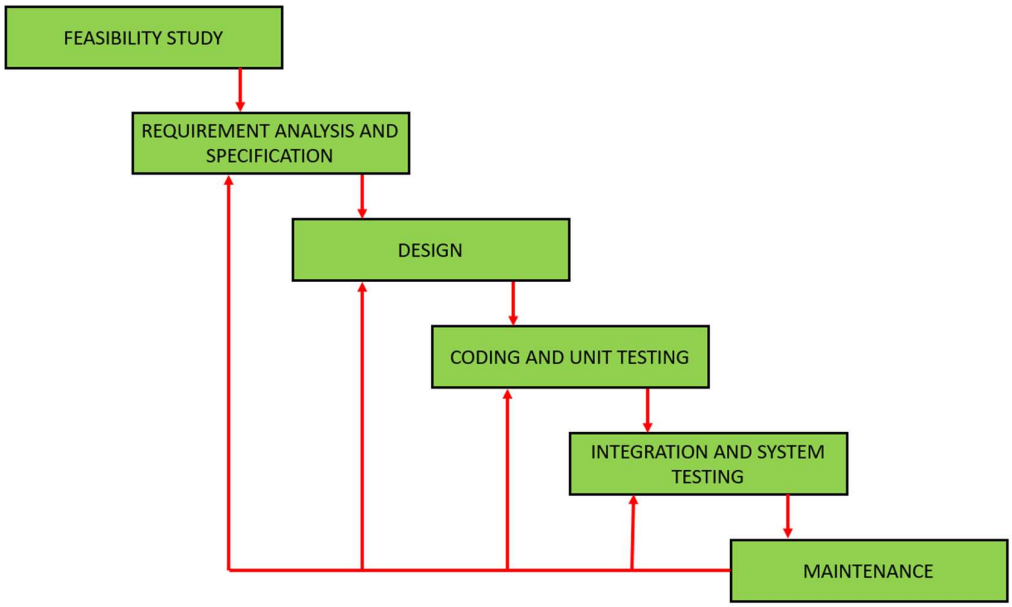
Software Engineering

Disadvantages of the Waterfall Model

Disadvantage	Explanation
1. Inflexible to Changes	Once the project moves to the next phase, changes are difficult and costly to implement.
2. Late Testing (High Risk of Bugs)	Testing happens at the end, which means bugs may be discovered late, making fixes expensive.
3. Not Suitable for Large Projects	If requirements are complex or change frequently, the Waterfall Model is inefficient.
4. No Customer Feedback During Development	Clients only see the final product, which may not meet their expectations.
5. High Risk & Uncertainty	If an error is found in an early phase, the entire project may need to be restarted.
6. Longer Time-to-Market	Since testing and deployment come at the end, the software takes longer to reach users.

Software Engineering

Iterative WaterFall Model –



Key Features -

1. Phase-by-Phase Feedback and Refinement

2. Controlled Iterations - allows limited revisions between specific phases.

3. Testing at Multiple Stages

4. Clear Ptoject Timeline

5. Customer Feedback is Considered - Customers or stakeholders review each iteration and provide feedback before final development.

6. Best for Large & Complex Projects

Advantages of the Iterative Waterfall Model

- **Allows Mid-Phase Corrections – Issues can be fixed within iterations before moving forward.**

Software Engineering

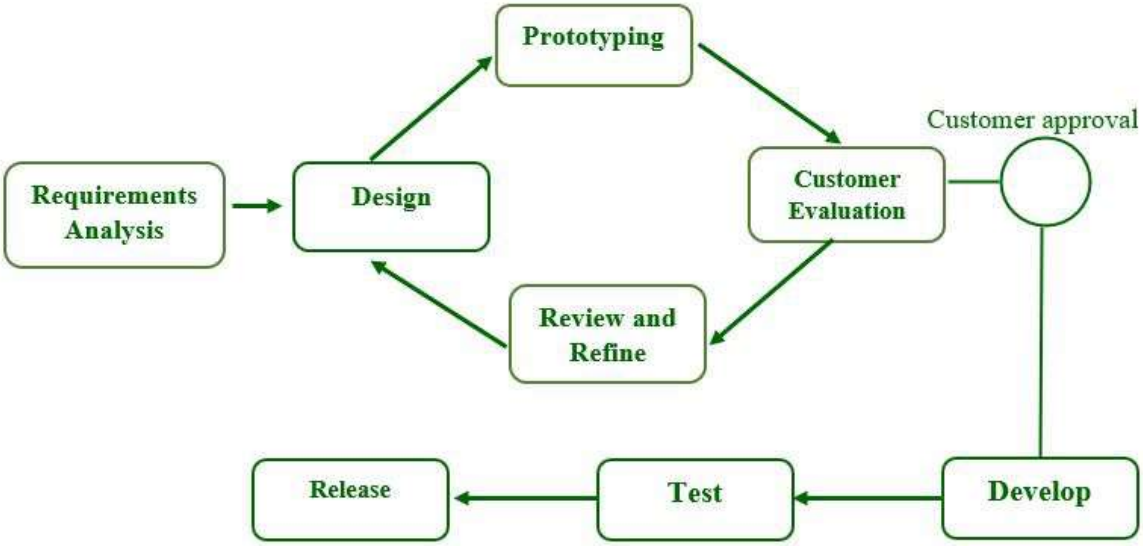
- **Early Bug Detection – Testing happens at each iteration, reducing final-stage errors.**
- **Customer Involvement – Clients can review and suggest changes at different stages.**
- **Lower Risk – Incremental improvements prevent major failures at later stages.**
- **Cost-Effective in Long Term – Fixing issues early reduces overall project cost.**
- **Better Software Quality – Iterative refinements improve the final product.**
- **Suitable for Large Projects – Works well when requirements evolve but need a structured approach.**

Disadvantages of the Iterative Waterfall Model

- **Slower Development – Multiple iterations extend the project timeline.**
- **Higher Costs – Additional testing and refinements require more resources.**
- **Increased Documentation – Each iteration needs proper documentation.**
- **Not as Flexible as Agile – Limited adaptability compared to Agile methodologies.**
- **Complex Project Management – Handling multiple iterations requires skilled developers.**

- **Limited customer interaction**
- **Risk handling not supported**

Prototype Model -



Steps of the Prototyping Model

1. Requirement Gathering and Analysis
    - Users express their needs, and a high-level understanding of what they want is gathered from the customers.
2. Quick Design
    - A basic design is created that includes only the essential features for the initial prototype. This serves as an outline for further development.
3. Build a Prototype
    - The initial prototype is developed based on the quick design. It includes only the basic functionalities that the customer requested.

4. Initial User Evaluation
    - Customers review the prototype, and feedback is gathered regarding its functionality, strengths, and weaknesses.
5. Refining the Prototype
    - The prototype is refined by addressing user feedback, and the design is adjusted accordingly. This process repeats until the customer is satisfied.
6. Implement Product and Maintain
    - After user approval, the prototype is turned into the final product. It is then implemented and maintained.

Types of Prototyping Models

1. Rapid Throwaway Prototyping
    - The prototype is discarded after feedback is gathered, and a new prototype is built from scratch. This method allows for quick exploration of ideas without much focus on maintaining the model for future iterations.
2. Evolutionary Prototyping
    - The prototype is developed incrementally. Each version of the prototype is built upon based on user feedback until it meets the required standards. This approach is more time-efficient than starting from scratch with each iteration.
3. Incremental Prototyping
    - The final product is broken into smaller parts, and each part is prototyped and tested individually. Once all the components are developed, they are merged to form the complete product.

4. Extreme Prototyping
    - Primarily used in web development, this method involves three phases:
        - Phase 1: Basic prototypes with static pages in HTML.
        - Phase 2: Functional screens with simulated data.
        - Phase 3: Final implementation of all services and functionalities.

Advantages of the Prototyping Model

- Early visibility of partial product increases customer satisfaction.
- Easy accommodation of new requirements.
- Missing features can be identified early.
- Early detection of errors saves time and cost.
- Prototype can be reused in future projects.
- Flexibility in design changes.
- Early customer feedback ensures the product meets expectations.
- Helps validate design decisions before full development.
- Reduces project risk by identifying issues early.
- Improves collaboration between team members and stakeholders.
- Bridges the gap between technical and non-technical stakeholders.

Disadvantages of the Prototyping Model

- High costs in terms of time and money.
- Frequent requirement changes lead to variation.

- Poor documentation due to constant changes.
- Difficult to accommodate all customer changes.
- Uncertainty in the number of iterations needed.
- Customers may expect the final product prematurely.
- Prototypes may lead to sub-optimal solutions.
- Customer disinterest if the initial prototype is unsatisfactory.
- Prototypes may not be scalable for future needs.
- Prototypes may not fully represent the final product.
- Focus on prototypes may delay final product development.
- False sense of completion could lead to premature release.
- Technical feasibility may be overlooked.
- Mismatch in tools and technologies used for prototypes.
- Prototypes may not align with business requirements.

Incremental Model –

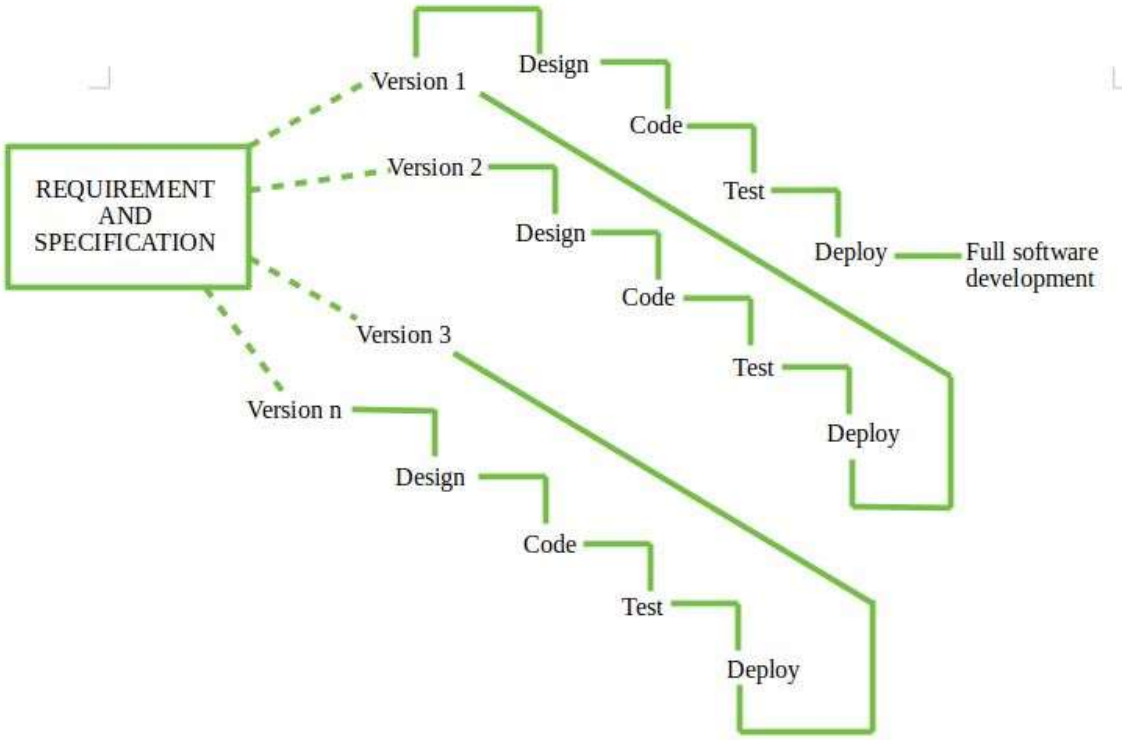
The Incremental Process Model is also known as the Successive version model . The **Incremental Model** is a method of software development where the system is designed, implemented, and tested incrementally (in parts or modules) until the product is complete. It breaks down the project into smaller, manageable components that are developed and refined over several iterations or increments.

Key Features of the Incremental Model

- **Modular Approach:** The product is broken down into smaller, manageable components (increments) that are developed and delivered in phases.

Software Engineering

- **Sequential Delivery:** Each increment is delivered sequentially with each iteration adding more functionality until the final system is complete.
- **Customer Feedback:** After each increment, the customer reviews the product and provides feedback to help refine future iterations.
- **Parallel Development:** The increments can be developed concurrently, speeding up the process.



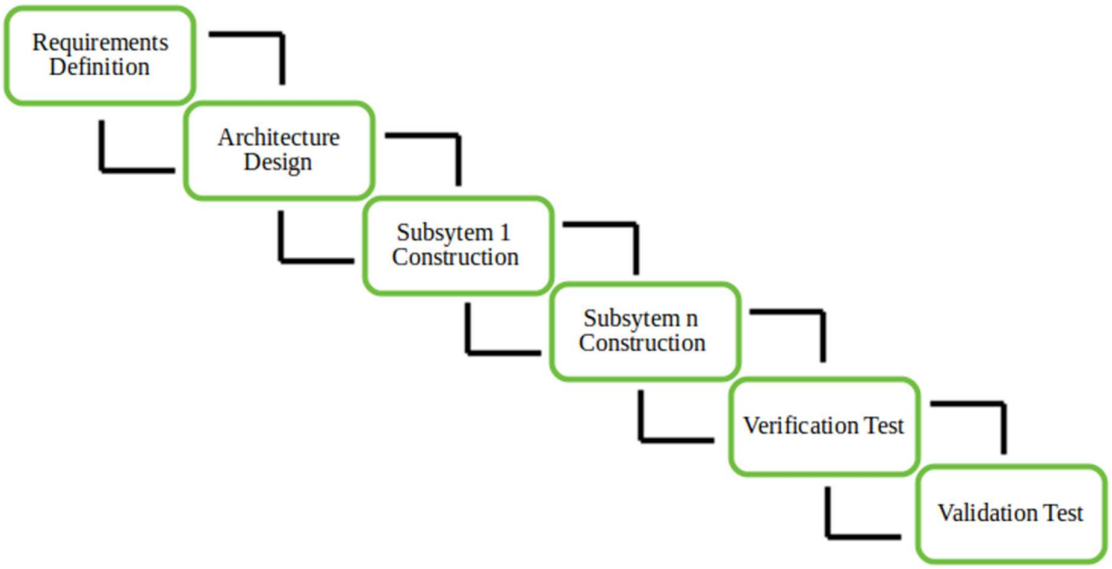
Types of Incremental Model

1. Staged Delivery Model
2. Parallel Development Model

Software Engineering

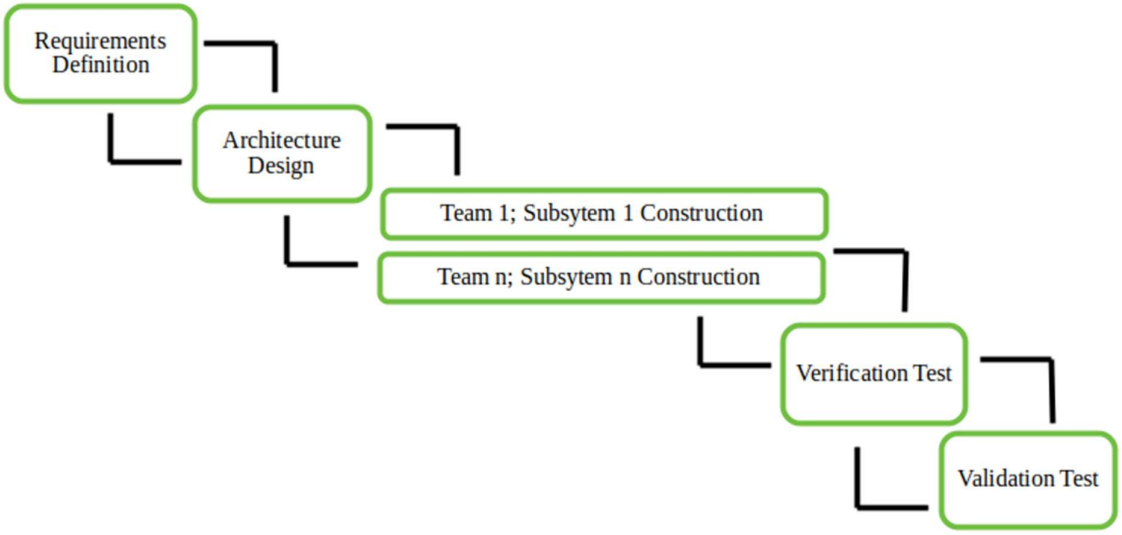
1. Staged Delivery Model

Construction of only one part of the project at a time.



2. Parallel Development Model

Different subsystems are developed at the same time. It can decrease the calendar time needed for the development, i.e. TTM (Time to Market) if enough resources are available.



Software Engineering

When to use the Incremental Process Model

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects have lengthy development schedules.
4. Projects with new Technology.
    - Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly).
    - Uses divide and conquer for a breakdown of tasks.
    - Lowers initial delivery cost.
    - Incremental Resource Deployment.
5. Requires good planning and design.
6. The total cost is not lower.
7. Well-defined module interfaces are required.

Advantages of the Incremental Process Model

1. Prepares the software fast.
2. Clients have a clear idea of the project.
3. Changes are easy to implement.
4. Provides risk handling support, because of its iterations.
5. Adjusting the criteria and scope is flexible and less costly.
6. Comparing this model to others, it is less expensive.
7. The identification of errors is simple.

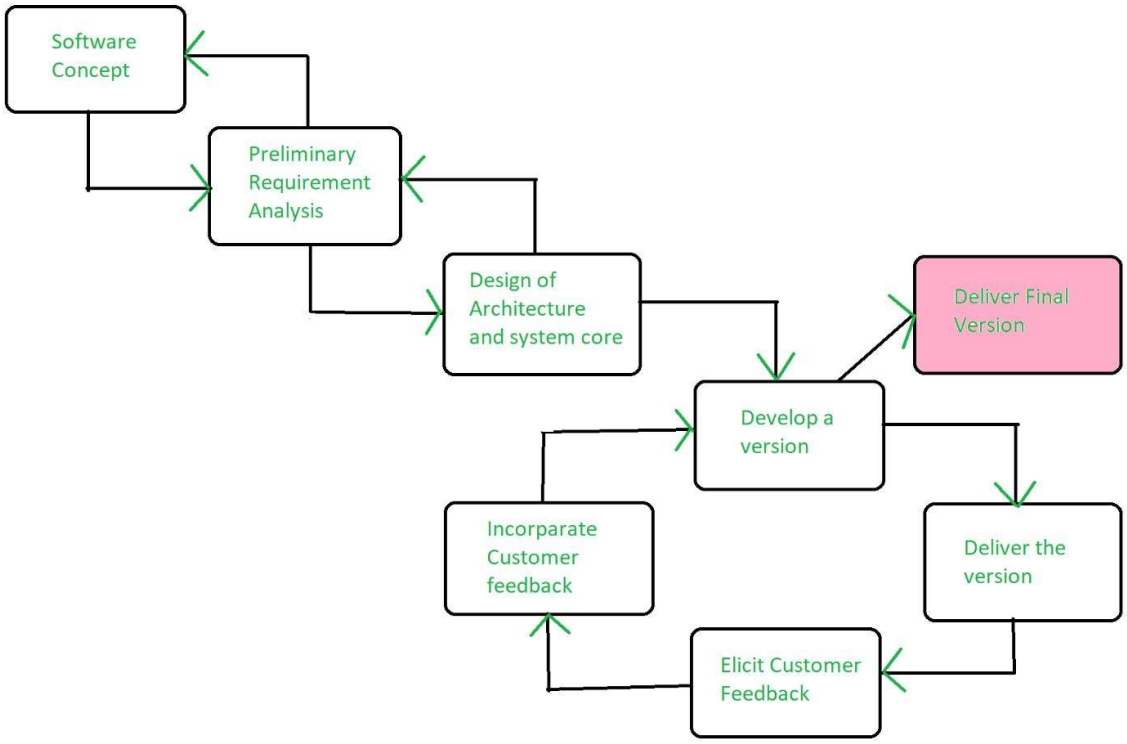
Software Engineering

Disadvantages of the Incremental Process Model

1. A good team and proper planned execution are required.
2. Because of its continuous iterations the cost increases.
3. Issues may arise from the system design if all needs are not gathered upfront throughout the program lifecycle.
4. Every iteration step is distinct and does not flow into the next.
5. It takes a lot of time and effort to fix an issue in one unit if it needs to be corrected in all the units.

Evolutionary Model -

The evolutionary model is a combination of the iterative and Incremental models of the software development life cycle.



Software Engineering

Key Features of the Evolutionary Model

- **Iterative Development:** The system is developed in repeated cycles (iterations), and each iteration produces a version of the system that is tested and refined.
- **Continuous Refinement:** The system evolves by continuously refining the product based on feedback and new requirements that emerge over time.
- **Prototyping:** The initial version of the system may act as a prototype, and subsequent versions are developed based on the prototype's performance and user feedback.
- **Flexible and Adaptive:** This model is highly flexible, allowing for changes in requirements even after the development process begins.

Steps in the Evolutionary Model

1. **Requirement Gathering and Analysis:** Initial requirements are gathered, but they are kept flexible to allow changes as development progresses.
2. **Preliminary Design:** A basic design of the system is created that can evolve over iterations.
3. **Build the First Version:** A working prototype or version of the system is developed based on the preliminary design.
4. **User Evaluation:** The user evaluates the first version of the system and provides feedback.
5. **Refinement and Iteration:** Based on the user feedback, the system is refined, and new iterations are built to improve the functionality and address any changes or new requirements.

6. **Final Product:** After several iterations, the system reaches a stable state and is delivered as the final product.

Necessary Conditions for Implementing this Model

1. Customer needs are clear and been explained in deep to the developer team.
2. There might be small changes required in separate parts but not a major change.
3. As it requires time, so there must be some time left for the market constraints.
4. Risk is high and continuous targets to achieve and report to customer repeatedly.
5. It is used when working on a technology is new and requires time to learn.

Advantages of the Evolutionary Model

- **Flexibility and Adaptability:** The model allows changes to be incorporated at any stage of development, making it very adaptive to changing requirements.
- **Early Delivery of Functional System:** A functional system (even if not complete) can be delivered early, providing a working version for the users to evaluate.
- **User Involvement:** Continuous user feedback ensures that the final product closely aligns with user needs.
- **Risk Management:** Problems or changes are identified early in the development process, reducing the risk of failure.
- **Reduced Complexity:** Initial versions are simple, allowing the team to focus on core functionalities before adding complexity.

Software Engineering

Disadvantages of the Evolutionary Model

- **Unclear End Product:** Since the system evolves with each iteration, the end product may not be clearly defined from the start, leading to scope creep.
- **Requires Strong Management:** Continuous changes can lead to difficulties in managing the project, and without clear goals, the project may drift.
- **Increased Cost:** Since changes are made continuously, the cost can increase due to the need for repeated testing and development.
- **Difficulty in Planning:** Estimating time and resources can be difficult due to the evolving nature of the project.
- **Excessive Refinement:** There is a risk of over-refinement and constant changes that can delay the final product and lead to inefficiency.

When to Use the Evolutionary Model

- **Complex Systems:** When developing complex systems where the requirements are not fully understood at the beginning or are likely to change frequently.
- **Prototyping Needs:** If there's a need for iterative prototyping and constant user feedback.
- **Highly Dynamic Requirements:** Suitable when the customer is uncertain about their needs and requires ongoing refinements throughout the development cycle.
- Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Software Engineering