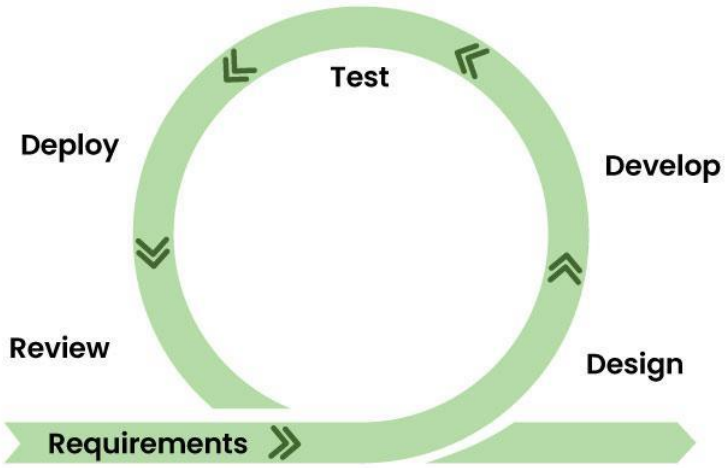
Introduction to Software Engineering

Q. Explain Agile Process Model

The **Agile Process Model** is a **modern software development approach** that focuses on **flexibility, customer collaboration, and incremental delivery**. It allows teams to adapt to changes **quickly and efficiently** compared to traditional models like the **Waterfall Model**.



Agile Model

Agile is a **lightweight, iterative, and incremental model** used for software development. It follows the **Agile Manifesto (2001)**, which emphasizes:

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a fixed plan

Instead of developing the entire software in one go, Agile delivers the software in **small, functional parts called iterations**. Each iteration typically lasts **1-4 weeks** and includes planning, development, testing, and feedback.

Phases of the Agile Process Model

The Agile model consists of several iterative cycles, each having the following phases:

1. Concept / Requirement Gathering

- Requirements are gathered from the client but are not fixed.

- Developers work closely with the client to understand high-level needs.

2. Iteration / Incremental Planning

- The entire project is divided into **small iterations**.
- Each iteration is planned separately based on priority.

3. Design & Development

- Developers start coding based on requirements.
- Since Agile is iterative, only **a small part of the software** is developed at a time.

4. Testing

- After development, the iteration is tested using **automated and manual testing**.
- Frequent feedback helps in finding and fixing defects early.

5. Release & Deployment

- After testing, a **functional version of the software** is delivered to the customer.
- The customer provides feedback, and the next iteration is planned accordingly.

6. Feedback & Continuous Improvement

- Customer and team members analyze the results.
- The next iteration is refined based on feedback, ensuring **continuous improvement**.

Agile Methodologies

There are several methodologies based on Agile principles:

1. Scrum

- One of the most popular Agile frameworks.
- Work is divided into **Sprints (1-4 weeks)**.
- **Daily Stand-up Meetings** help track progress.
- Uses roles like **Scrum Master, Product Owner, and Development Team**.

2. Kanban

- Uses a **visual board** to track the progress of tasks.
- Helps in limiting work-in-progress (WIP) to **increase efficiency**.
- Focuses on **continuous delivery** without fixed sprints.

3. Extreme Programming (XP)

- Focuses on **continuous testing and frequent releases**.

- Uses **Pair Programming** where two developers code together.
- Encourages high customer involvement.

Advantages of Agile

✓ **Faster Delivery** → Software is developed in short cycles.
✓ **Customer Satisfaction** → Regular feedback ensures the final product meets expectations.
✓ **Flexibility** → Changes can be incorporated easily.
✓ **Better Risk Management** → Issues are identified early due to continuous testing.
✓ **Improved Collaboration** → Developers, testers, and stakeholders work together.

Disadvantages of Agile

✓ **Requires Active Customer Involvement** → Continuous feedback is needed.
✓ **Difficult to Scale for Large Teams** → Managing multiple Agile teams is complex.
✓ **Lack of Proper Documentation** → Focus is on working software rather than documentation.
✓ **Frequent Changes Can Cause Delays** → If requirements keep changing, the project may extend beyond the expected timeline.

Q. Explain Agile Principles

The **Agile Manifesto**, introduced in **2001**, defines **12 Agile Principles** that guide Agile software development. These principles focus on **customer satisfaction, flexibility, teamwork, and continuous improvement**.

---

12 Agile Principles Explained

1. Customer Satisfaction Through Early and Continuous Delivery

- The highest priority is to **deliver valuable software** early and frequently.
- Continuous delivery keeps customers engaged and satisfied.

Example: Instead of waiting for months to release software, Agile delivers small, working versions every few weeks.

---

2. Welcome Changing Requirements, Even Late in Development

- Agile embraces **change at any stage** to meet business needs.

- Unlike traditional models (e.g., Waterfall), Agile allows modifications throughout the project.

Example: If a client wants a new feature **midway through development**, Agile teams can add it in the next iteration.

---

3. Deliver Working Software Frequently

- Agile follows an **iterative approach**, delivering software in short cycles (1-4 weeks).
- Frequent delivery reduces risks and improves feedback loops.

Example: Instead of releasing an entire e-commerce website at once, Agile teams first launch a **basic version with essential features**, then add more features over time.

---

4. Collaboration Between Business and Development Teams

- Continuous communication between **developers and stakeholders** is crucial.
- Direct interaction ensures **clear understanding of requirements** and prevents misunderstandings.

Example: Developers work closely with a **bank's business team** to build an online banking system that meets customer needs.

---

5. Build Projects Around Motivated Individuals

- Teams should be **self-organizing and motivated** to take ownership of their work.
- Give them **freedom and trust**, instead of strict micromanagement.

Example: A startup allows developers to choose their **own tasks** and encourages creativity in problem-solving.

---

6. Face-to-Face Communication is Most Effective

- Agile prefers **direct communication** over emails or long documents.
- Daily stand-up meetings help in quick decision-making.

Example: In Scrum, teams hold **daily stand-up meetings** to discuss progress instead of writing long email updates.

---

7. Working Software is the Primary Measure of Progress

- The success of a project is measured by **functional software**, not documents or reports.

- Focus is on **delivering usable features** in each iteration.

Example: A **partially working app** is more valuable than a **detailed 100-page project plan** without actual code.

---

8. Maintain a Sustainable Development Pace

- Teams should work at a **consistent, manageable speed** without overworking.
- Avoiding burnout ensures **higher productivity and quality**.

Example: Instead of working **60+ hours per week** to meet unrealistic deadlines, Agile teams follow a balanced work schedule.

---

9. Continuous Attention to Technical Excellence and Good Design

- Agile promotes **clean coding practices, code reviews, and refactoring**.
- High-quality design ensures long-term maintainability.

Example: Using **design patterns** like MVC (Model-View-Controller) to keep code structured and efficient.

---

10. Simplicity – The Art of Maximizing Work Not Done

- Focus on **essential tasks only**, avoiding unnecessary complexity.
- Simple solutions are easier to implement, test, and maintain.

Example: Instead of building a **complex payment system**, a startup integrates a **third-party payment gateway** (like Razorpay or Stripe) to save time.

---

11. Self-Organizing Teams Produce the Best Results

- Agile teams **make their own decisions** rather than relying on a strict hierarchy.
- Encourages innovation and responsibility.

Example: A team decides which **features to prioritize** based on customer feedback, without waiting for top management approval.

---

12. Regular Reflection and Adjustment

- Teams regularly review their work and **identify areas for improvement**.
- Retrospectives help refine the Agile process.

Example: After every sprint, the team holds a **retrospective meeting** to discuss what went well and what needs improvement.

Q. Write short note on: SCRUM

Scrum is an **Agile framework** used for software development and project management. It focuses on **iterative development, teamwork, and flexibility**. Scrum helps teams deliver high-quality software **in small, incremental steps** instead of waiting for the entire project to be completed.

Unlike traditional models like the **Waterfall Model**, where development follows a **fixed sequence (planning → design → development → testing → deployment)**, Scrum follows **Sprints (short cycles of 1-4 weeks)** to develop and deliver software **frequently** with continuous feedback.

Scrum Framework

Scrum consists of **Roles, Events, and Artifacts** that work together to ensure smooth project execution.

A. Scrum Roles (Who is involved?)

Scrum has three main roles:

1. **Scrum Master**
    - Ensures that the team follows Scrum principles.
    - Removes obstacles that slow down the team.
    - Facilitates Scrum meetings and discussions.
2. **Product Owner**
    - Represents the **customer and business needs**.
    - Maintains the **Product Backlog** (list of tasks & features).
    - Prioritizes work to maximize project value.
3. **Development Team**
    - A **self-organizing team** responsible for coding, testing, and delivering the software.
    - Typically includes developers, testers, and designers.

B. Scrum Events (What happens in Scrum?)

Scrum follows a set of well-defined **meetings/events** to ensure smooth execution.

1. **Sprint**
    - A short development cycle, usually **1-4 weeks** long.
    - The goal is to complete a **working feature** in each Sprint.
    - The work for a Sprint is taken from the **Sprint Backlog**.
2. **Sprint Planning**
    - A meeting before the Sprint begins.
    - The team selects and plans the tasks they will complete in the Sprint.
3. **Daily Stand-up (Daily Scrum)**
    - A **15-minute meeting** held every day.
    - Team members discuss:
      ✓ What they did yesterday
      ✓ What they will do today
      ✓ Any obstacles they face
4. **Sprint Review**
    - Held at the end of the Sprint.
    - The team presents the completed work to the **Product Owner & stakeholders**.
5. **Sprint Retrospective**
    - A meeting to **reflect** on what went well and what needs improvement.
    - Helps the team improve their work process for the next Sprint.

C. Scrum Artifacts (What is used in Scrum?)

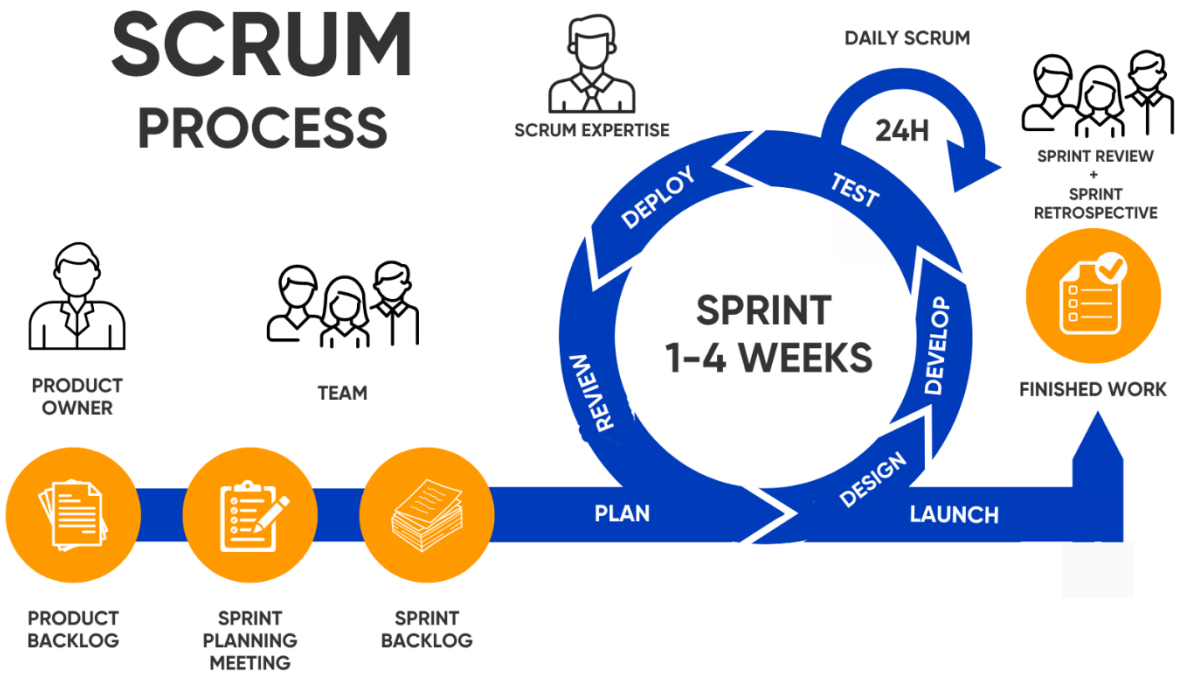
Scrum uses three main artifacts to manage work efficiently:

1. **Product Backlog**
    - A **list of all features, tasks, and improvements** needed in the project.
    - Managed by the **Product Owner**.
    - Continuously updated as per project needs.
2. **Sprint Backlog**
    - A subset of the **Product Backlog**, containing tasks selected for the current Sprint.

- The **Development Team** decides what they can complete within the Sprint.

3. **Increment**
    - The **final working product** delivered at the end of a Sprint.
    - It is a small, functional part of the overall software.

Rabiloo



Scrum Workflow (How does Scrum work?)

1. **Product Owner creates a Product Backlog** (list of all required features).
2. **Sprint Planning** → The team selects tasks from the backlog.
3. **Sprint starts (1-4 weeks)** → The team works on selected tasks.
4. **Daily Stand-up** → Team discusses progress daily.
5. **Sprint Review** → Completed features are demonstrated.
6. **Sprint Retrospective** → The team reflects on improvements.
7. **Next Sprint begins** → The process repeats until the product is fully developed.

Advantages of Scrum

✓ **Faster delivery** of working software.
✓ **Easily adaptable** to changing requirements.
✓ **Improves collaboration** between team members.
✓ **Early issue detection** through daily stand-ups.
✓ **Customer satisfaction** due to continuous feedback.

Disadvantages of Scrum

✓ **Requires a disciplined team** for smooth execution.
✓ **Frequent meetings can be time-consuming**.
✓ **Not suitable for large teams** or very complex projects.
✓ **Changing requirements may cause scope creep** (project going beyond original goals).

Q. Explain Kanban Model

The **Kanban Model** is an **Agile framework** used for **workflow management** and **continuous delivery** of software. It helps teams **visualize work, limit work-in-progress (WIP), and improve efficiency**.

Kanban was originally developed in **Toyota's manufacturing system** to manage inventory and production. Later, it was adapted for **software development** to improve workflow efficiency.

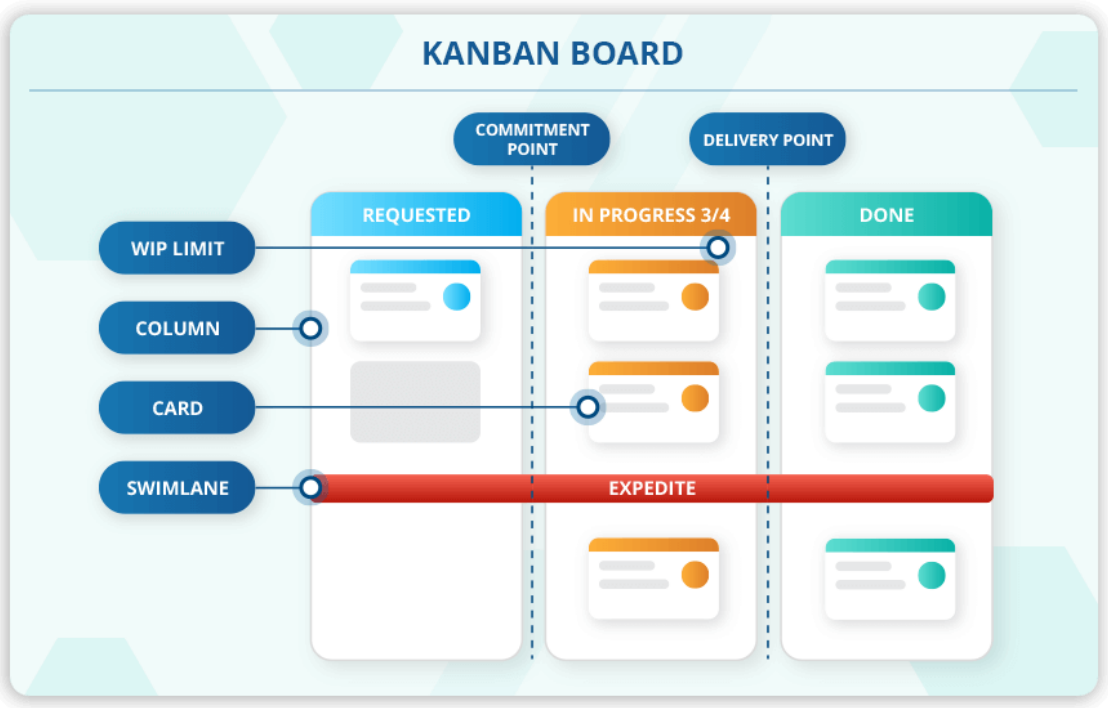
Key Principles of Kanban

Kanban follows **six principles**:

1. **Visualize the Workflow**
    - Uses a **Kanban Board** with columns like **To Do, In Progress, and Done** to track tasks.
2. **Limit Work in Progress (WIP)**
    - Restricts the number of tasks in progress at any time to avoid overload.
3. **Manage Flow**
    - Focuses on smooth and continuous task completion to **reduce delays**.
4. **Make Process Policies Explicit**
    - Defines clear rules for moving tasks between workflow stages.
5. **Implement Feedback Loops**

- Regular review meetings help improve the workflow.

6. **Collaborate for Continuous Improvement**
   - Teams analyze workflow to **optimize efficiency** and eliminate bottlenecks.



Kanban Board

The **Kanban Board** is a visual tool used to track tasks and progress. It has **three main sections**:

To Do In Progress Done

Task 1 Task 4 Task 6

Task 2 Task 5 Task 7

Task 3

- **New tasks** are added in the **To Do** column.
- As work begins, tasks move to **In Progress**.
- Once completed, they move to **Done**.

How Kanban Works?

1. **Create a Kanban Board** → Define columns based on workflow stages.
2. **Prioritize Tasks** → Add tasks to the **To Do** column.

3. **Set WIP Limits** → Restrict the number of tasks in progress.
4. **Work on Tasks** → Move tasks through the board.
5. **Monitor & Improve** → Regularly analyze and refine the workflow.

Advantages of Kanban Model

✓ **Simple & easy to implement**
✓ **Improves workflow visibility**
✓ **Reduces task overload** by limiting WIP
✓ **Faster delivery of work**
✓ **Continuous improvement of processes**

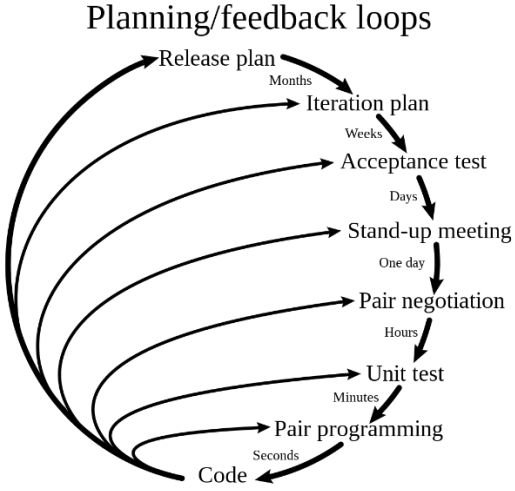
Disadvantages of Kanban Model

✓ **Difficult to manage in large teams**
✓ **Requires constant monitoring**
✓ **Not suitable for complex software development projects**

Q. Explain XP model

Extreme Programming (XP) is an **Agile software development methodology** that focuses on **high-quality code, frequent releases, and customer collaboration**. It was introduced by **Kent Beck in the 1990s** and is widely used for projects that require **continuous changes and quick delivery**.

XP is best suited for **small to medium-sized teams** that need to adapt to changing requirements quickly while ensuring **high code quality** through testing and teamwork.



Key Principles of XP Model

XP follows **five core principles** that guide the development process:

1. **Communication** → Encourages frequent discussions between team members and customers.
2. **Simplicity** → Focuses on writing only the necessary code, avoiding complexity.
3. **Feedback** → Developers receive continuous feedback through testing and customer reviews.
4. **Courage** → Encourages developers to take necessary risks, refactor code, and make improvements.
5. **Respect** → Team members value each other's contributions and work collaboratively.

Phases of XP Model

The XP development cycle consists of **six key phases**:

1. Planning

- Customer defines **user stories** (short descriptions of features).
- The team estimates the time required for each story.

2. Designing

- Uses **simple and flexible designs** to allow future modifications.
- Encourages the use of **design patterns** for better code organization.

3. Coding

- Developers work in **pairs (Pair Programming)** for better code quality.
- The code is continuously tested and reviewed.

4. Testing

- Follows **Test-Driven Development (TDD)** where tests are written **before writing the actual code**.
- Continuous Integration (CI) ensures that all code changes are tested automatically.

5. Integration

- Code is merged frequently (often multiple times a day) to avoid conflicts.

6. Deployment & Maintenance

- The software is deployed frequently in **small releases**.

- Customer feedback is used to refine and improve the system.

Key Practices of XP

XP follows **12 best practices** that improve the software development process:

1. **Pair Programming** → Two developers work on the same code to ensure better quality.
2. **Test-Driven Development (TDD)** → Write tests before writing code.
3. **Continuous Integration** → Code is merged and tested frequently.
4. **Small Releases** → Software is delivered in small, frequent updates.
5. **Simple Design** → Only essential features are implemented to avoid complexity.
6. **Refactoring** → Improve and clean up the code continuously.
7. **Collective Code Ownership** → The entire team can modify the code.
8. **Coding Standards** → Follow uniform coding rules for consistency.
9. **Metaphor** → Use simple, real-world terms to describe the system.
10. **Sustainable Pace** → Avoid overworking the team (no long overtime).
11. **On-Site Customer** → A customer representative is always available for feedback.
12. **System Testing** → Automated and manual testing is performed regularly.

Advantages of XP Model

✓ **High-quality software** due to continuous testing and pair programming.
✓ **Quick response to changes** in customer requirements.
✓ **Improved teamwork and communication**.
✓ **Early bug detection** through Test-Driven Development (TDD).
✓ **Faster delivery of working software**.

Disadvantages of XP Model

✓ **Requires experienced developers** for pair programming and TDD.
✓ **Frequent customer involvement** may not always be possible.
✓ **Not suitable for large teams** or complex projects.
✓ **Too many short iterations** can make planning difficult.

Q. Define Software. Describe the Characteristics of software

Software is a **collection of programs, data, and instructions** that tell a computer how to perform specific tasks. It is intangible and enables hardware to function efficiently.

Formal Definition:

"Software is a set of instructions, data, or programs used to operate computers and execute specific tasks."

Characteristics of Software

Software is different from hardware as it does not have a physical form. Here are its key characteristics:

1. Intangibility

- Software has no physical existence; it is a collection of **code and data** stored in a computer.
- Unlike hardware, it **cannot be touched or physically modified**.

2. Complexity

- Software development is **highly complex**, involving millions of lines of code.
- Even small programs require **detailed logic and design** to function correctly.

3. Flexibility

- Software can be **modified and updated** to fix errors or improve performance.
- Unlike hardware, it does not wear out but requires maintenance.

4. Reusability

- Software components can be **reused** in different applications.
- Example: A **login module** can be reused in multiple websites.

5. Maintainability

- Software needs **continuous updates, bug fixes, and improvements**.
- This is known as **Software Maintenance**, which includes enhancements and debugging.

6. Scalability

- Software can be **scaled up** to handle more users or complex operations.
- Example: Websites like Facebook and Amazon expand their software to support millions of users.

7. Portability

- Software can be **run on different hardware platforms** without major changes.
- Example: A **web browser** can work on Windows, Linux, and macOS.

8. Efficiency

- Good software consumes **minimum resources (CPU, memory, power)** while delivering high performance.
- Example: A well-optimized mobile app does not drain the battery quickly.

9. Reliability

- Software must work **consistently without failures**.
- Example: Banking software should not crash during transactions.

10. No Wear and Tear

- Unlike hardware, software does not degrade with use.
- However, it may become outdated and need **upgrades**.

Q. Explain Software Process Framework in Layered Approach

A **Software Process Framework** is a **structured approach** that defines the different stages involved in software development. It provides **guidelines, best practices, and workflows** to ensure efficient software creation and maintenance.

Layered Approach in Software Process Framework

The **Software Process Framework** follows a **layered approach**, which helps in organizing the different activities of software development in a systematic way.

Layers of Software Process Framework

The Software Process Framework consists of **four layers**:

1. Quality Focus (Core Layer)

✓ The foundation of the software process framework.
✓ Ensures that **quality assurance (QA) activities** are embedded in every stage of development.
✓ Involves **reviews, testing, audits, and process improvements** to maintain high standards.

2. Process Framework (Process Layer)

✓ Defines **how software development should be carried out**.
✓ It includes:

- **Software Development Life Cycle (SDLC)** (Waterfall, Agile, Spiral, etc.)
- **Process Models** (Incremental, Evolutionary, etc.)
  ✓ Acts as a **roadmap** for project execution.

3. Methods (Method Layer)

✓ Specifies **techniques and tools** used for software development.
✓ Includes:

- **Requirement Analysis Methods** (Use Cases, UML Diagrams)
- **Design Methods** (Architectural Patterns, UI/UX Design)
- **Coding Methods** (Programming Standards, Code Reviews)
- **Testing Methods** (Black Box, White Box, Automated Testing)

4. Tools (Tool Layer)

✓ Provides **software tools** to support different stages of development.
✓ Examples:

- **Requirement Gathering Tools** → JIRA, Trello
- **Design Tools** → UML Diagrams, Figma
- **Coding Tools** → IDEs (Eclipse, VS Code)
- **Testing Tools** → Selenium, JUnit
- **Project Management Tools** → MS Project, GitHub

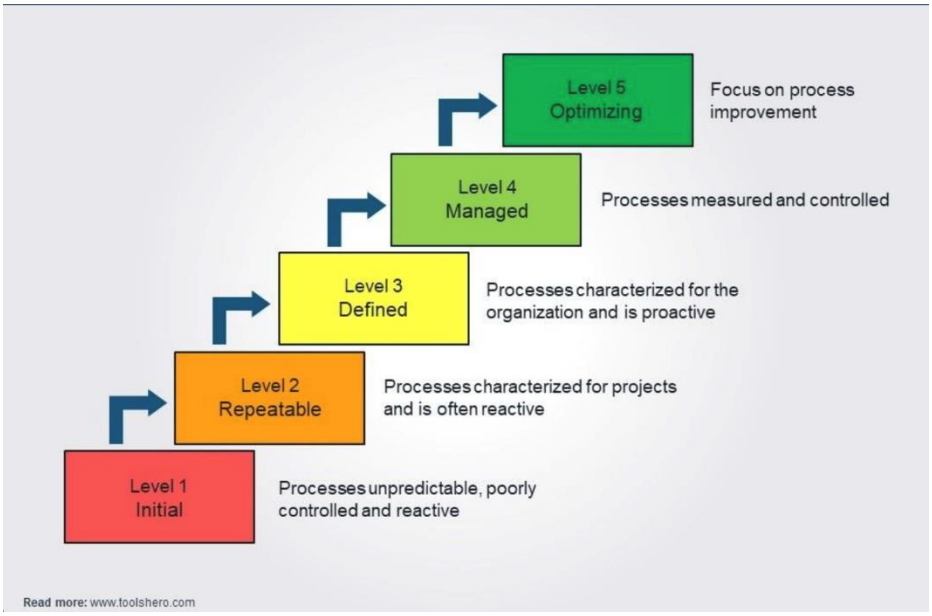
Advantages of the Layered Approach

✓ **Structured and Organized Development** → Ensures step-by-step execution.
✓ **Improves Software Quality** → Quality assurance is integrated at every layer.
✓ **Enhances Productivity** → Clearly defined processes and methods improve efficiency.
✓ **Facilitates Tool Support** → Automated tools enhance accuracy and reduce errors.
✓ **Flexible and Scalable** → Can be adapted for different project requirements.

Q. Explain Capability Maturity Model

The **Capability Maturity Model (CMM)** is a **framework** that helps organizations improve their **software development processes**. It provides a **structured path** for organizations to move from **ad-hoc and chaotic** software development to a **disciplined and mature** process.

CMM was developed by the **Software Engineering Institute (SEI)** at **Carnegie Mellon University** in the late 1980s.



CMM Structure and Purpose

✓ CMM defines **five levels** of process maturity.
✓ Each level represents an **improvement in the organization's ability** to develop high-quality software.
✓ The goal is to move from **Level 1 (Initial) → Level 5 (Optimizing)** to ensure **continuous process improvement**.

Five Maturity Levels of CMM

CMM is divided into **five maturity levels**, where each level describes how well an organization **manages its software development processes**.

Level 1: Initial (Chaotic)

✓ Processes are **unpredictable and poorly controlled**.
✓ Software development is done in an **ad-hoc manner**.
✓ Success depends on **individual effort**, not a defined process.
✓ **Challenges:** High risks, frequent project failures, poor quality.

Example: A startup developing software without any formal guidelines.

Level 2: Repeatable (Managed)

✓ Basic **project management** practices are established.
✓ **Processes are documented** so they can be repeated.
✓ Organizations follow **basic software engineering practices** like **version control and tracking defects**.
✓ **Challenges:** Still lacks well-defined processes across teams.

Example: A company starts using **Gantt charts, project tracking tools, and documentation**.

Level 3: Defined

✓ The **organization standardizes** its software development process.
✓ **Formal procedures, guidelines, and documentation** are established.
✓ Teams follow a **well-defined** process model such as **Agile or Waterfall**.
✓ **Challenges:** Processes exist, but optimization is needed.

Example: A company uses **Scrum methodology**, conducts **regular code reviews**, and follows **defined software development life cycle (SDLC)**.

Level 4: Managed (Quantitatively Controlled)

✓ **Quantitative measures** (metrics) are used to track **software quality and process performance**.
✓ Organizations use **data-driven decision-making**.
✓ **Defect rates, productivity, and performance** are measured.
✓ **Challenges:** Continuous improvement based on data insights.

Example: A company tracks **code efficiency, defect rates, and team productivity** to optimize software development.

Level 5: Optimizing (Continuous Improvement)

✓ Focus on **continuous process improvement**.
✓ Uses **advanced analytics, automation, and AI-driven tools**.
✓ Teams continuously **analyze and improve processes** to prevent defects before they occur.
✓ **Challenges:** Requires ongoing innovation and investment.

Example: Google uses **AI-based code review tools, DevOps automation, and continuous feedback loops** to improve software development.

Benefits of CMM

✓ **Improves Software Quality** → Structured approach reduces defects.
✓ **Enhances Productivity** → Well-defined processes lead to faster development.
✓ **Reduces Risk** → Project failures decrease with proper planning.
✓ **Enables Continuous Improvement** → Organizations focus on refining processes.
✓ **Better Project Management** → Tracking and measuring performance ensures on-time delivery.

Q. Explain Capability Maturity Model Integration

Capability Maturity Model Integration (**CMMI**) is a **process improvement framework** that helps organizations **enhance software quality, productivity, and efficiency**. It builds on the **Capability Maturity Model (CMM)** and integrates different process models into a **single standard framework**.

Developed by the **Software Engineering Institute (SEI) at Carnegie Mellon University**, CMMI is used in **software development, project management, and business process improvement**.

Purpose of CMMI

✓ Helps organizations **standardize and improve** their software development processes.
✓ Provides a **structured roadmap** for continuous process improvement.
✓ Reduces **risks, defects, and inefficiencies** in software projects.

CMMI is widely used in **IT, defense, healthcare, finance, and manufacturing industries** to ensure **high-quality products and services**.

CMMI Maturity Levels

CMMI consists of **five maturity levels**, similar to CMM but with more focus on **integration, consistency, and process optimization**.

Level 1: Initial (Unpredictable & Chaotic)

✓ No structured process; **ad-hoc** and reactive.
✓ **High risks, frequent failures, and poor quality**.
✓ No proper **documentation or tracking** of project progress.

Example: A startup develops software without any formal guidelines or planning.

Level 2: Managed (Project-Level Control)

✓ Basic **project management** practices are established.
✓ Projects follow **documented procedures** but are not organization-wide.
✓ **Tracking and monitoring** of project schedules and costs.

Example: A company starts using **project management tools (JIRA, Trello)** and basic documentation.

Level 3: Defined (Organization-Wide Standards)

✓ **Standardized processes and policies** across the organization.
✓ Teams follow **structured methodologies** like Agile, Waterfall, or DevOps.
✓ **Process improvement** is based on **lessons learned from past projects**.

Example: A company adopts **Scrum methodology, regular code reviews, and defined development workflows**.

Level 4: Quantitatively Managed (Data-Driven Decisions)

✓ Organizations use **quantitative metrics** to **measure and control** process quality.
✓ Focus on **defect prevention, cost efficiency, and risk management**.
✓ Real-time **performance tracking** to optimize workflows.

Example: A company uses **KPIs (Key Performance Indicators), defect density, and automated quality assurance tools** to monitor software quality.

Level 5: Optimizing (Continuous Process Improvement)

✓ Focus on **continuous innovation and optimization**.
✓ Uses **automation, AI-driven insights, and predictive analytics** for improvement.
✓ **Processes are refined and updated continuously** to improve efficiency.

Example: Google, Amazon, and Microsoft use **AI-based development tools, continuous integration (CI/CD), and DevOps automation** for process improvement.

Key Differences Between CMM and CMMI

Feature	**CMM (Capability Maturity Model)**	**CMMI (Capability Maturity Model Integration)**
Focus	Only on **software development**	Covers **software + other processes** (management, services, etc.)

Feature	CMM (Capability Maturity Model)	CMMI (Capability Maturity Model Integration)
Structure	Separate models for different domains	**Integrated approach** for all processes
Flexibility	Less adaptable	More flexible and **customizable**
Measurement	Basic process measurement	**Advanced metrics and performance tracking**
Process Improvement	Not continuous	Focuses on **continuous process optimization**

Benefits of CMMI

✓ **Improves Software Quality** → Ensures **standardized** development processes.
✓ **Reduces Risks** → Helps in **early defect detection** and cost management.
✓ **Enhances Productivity** → Well-defined processes increase **efficiency**.
✓ **Better Decision-Making** → Uses **quantitative data** for process control.
✓ **Competitive Advantage** → Organizations with **CMMI certification** gain more trust from clients.

Q. Explain Prescriptive Model

A **Prescriptive Process Model** defines a **structured and systematic approach** for software development. It provides **clear guidelines, phases, and workflows** to ensure software is developed efficiently, with minimal errors.

These models are called **prescriptive** because they **prescribe** (i.e., dictate) **a specific way** to carry out software development.

Types of Prescriptive Process Models

Model	Approach	Pros	Cons	Best For
Waterfall	Linear & Sequential	Simple, Structured	No flexibility	Small projects with fixed requirements
V-Model	Testing at every phase	Early defect detection	Rigid, Expensive	Safety-critical software

Model	Approach	Pros	Cons	Best For
Incremental	Develop in increments	Early delivery, Flexible	Requires planning	Web & mobile applications
Prototyping	Build prototype first	Reduces misunderstanding	Can lead to scope creep	UI/UX-heavy apps
Spiral	Iterative with risk management	Best for high-risk projects	Costly & complex	Defense, aerospace

---

Advantages of Prescriptive Models

✓ **Clear structure and guidelines** → Easy to follow.
✓ **Ensures quality and consistency** → Well-defined phases reduce errors.
✓ **Better project tracking and management** → Defined milestones help in planning.
✓ **Good for projects with stable requirements** → Waterfall & V-model work well.

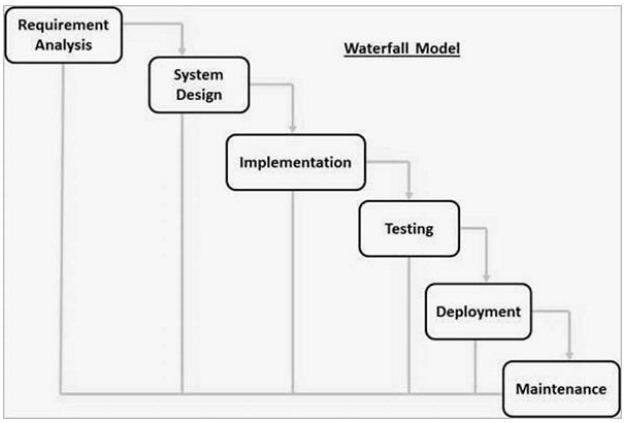
Disadvantages of Prescriptive Models

✓ **Less flexibility** → Not suitable for projects with frequently changing requirements.
✓ **High initial planning effort** → Requires detailed documentation.
✓ **May lead to delays** → If a problem is found in later stages, fixing it is costly.

Q. Explain Waterfall Model

The **Waterfall Model** is one of the **earliest and most traditional** software development models. It follows a **linear and sequential** approach, where each phase must be completed before moving to the next.

It is called a **"Waterfall"** model because the progress **flows downwards** like a **waterfall**, meaning once a phase is completed, you cannot go back to a previous phase easily.



Phases of the Waterfall Model

The Waterfall Model consists of **six main phases**:

1. Requirement Gathering & Analysis

✓ In this phase, all **functional and non-functional** requirements of the software are collected.
✓ A **Software Requirement Specification (SRS)** document is created.

Example: A bank wants an online banking system, so all features like login, fund transfer, and account statements are defined.

---

2. System Design

✓ Based on the requirements, the **architecture and design** of the software are planned.
✓ **Two types of designs are created:**

- **High-Level Design (HLD)** → Overall system architecture.
- **Low-Level Design (LLD)** → Detailed design of each module.

Example: In an e-commerce website, this phase defines how the shopping cart, checkout, and payment gateways will work together.

---

3. Implementation (Coding)

✓ Developers write **code** based on the design documents.
✓ Each module is developed separately and then integrated.

Example: Programmers write **Python, Java, or C++** code for different functions of a mobile banking app.

---

4. Testing

✓ The developed software is **tested** for bugs, errors, and defects.
✓ Different types of testing are performed:

- **Unit Testing** → Tests individual components.
- **Integration Testing** → Tests module interaction.
- **System Testing** → Tests the entire system.
- **Acceptance Testing** → Ensures software meets user expectations.

Example: In a hospital management system, testers check if patient records are correctly stored and retrieved.

---

5. Deployment

✓ Once testing is complete, the software is **deployed** in the real environment for end-users.

Example: A banking system is installed on **customer service desks and mobile apps**.

---

6. Maintenance

✓ After deployment, regular **updates and bug fixes** are done.
✓ Maintenance includes:

- **Corrective Maintenance** → Fixing bugs.
- **Adaptive Maintenance** → Updating for new OS or hardware.
- **Perfective Maintenance** → Improving features based on user feedback.

Example: A mobile banking app releases **new security updates** every 6 months.

Advantages of the Waterfall Model

✓ **Simple and easy to understand** → Each phase has clear goals.
✓ **Well-documented process** → Every phase is documented properly.
✓ **Easy project management** → Phases are well-structured, making it easy to manage large teams.
✓ **Best for fixed requirements** → Works well when software requirements **don't change**.

Disadvantages of the Waterfall Model

✓ **No flexibility** → Once a phase is completed, going back to make changes is difficult.
✓ **Late testing** → Bugs are only found after the coding phase, making them expensive to fix.
✓ **Not suitable for evolving projects** → If requirements change, the entire process must restart.
✓ **High risk & uncertainty** → If initial requirements are wrong, the whole project fails.

When to Use the Waterfall Model?

✓ **For small projects** with well-defined and **stable** requirements.
✓ **For projects with clear documentation** (e.g., **government or military software**).
✓ **When following strict regulations** (e.g., **banking, healthcare, or aerospace**).

When NOT to Use the Waterfall Model?

✓ **For complex projects** where requirements **may change** over time.
✓ **For Agile development** where customer feedback is required at every stage.
✓ **For startups** where quick prototyping and testing are needed.

Q. Explain V-Model

The **V-Model (Verification and Validation Model)** is an **extension of the Waterfall Model** where each development phase has a **corresponding testing phase**. It is called the **V-Model** because its shape looks like the letter **"V"**, showing the relationship between **development** and **testing**.

It is also known as the **Verification and Validation Model**, where:

- **Verification** → Ensures the software is built correctly (meets design specifications).
- **Validation** → Ensures the correct software is built (meets customer needs).

Phases of the V-Model

The V-Model consists of two sides:
✓ **Left Side (Verification – Development Phases)**
✓ **Right Side (Validation – Testing Phases)**

Left Side: Verification (Development Phases)

These phases involve **planning and designing** the software before coding starts.

1. Requirement Analysis
✓ Gather **functional & non-functional requirements** from users.
✓ Create a **Software Requirement Specification (SRS)** document.

Example: A banking app should allow users to transfer money, check balances, and receive notifications.

---

2. System Design
✓ Defines **overall architecture** and **hardware/software requirements**.

Example: Decide if the banking app should work on **cloud servers** or a **local database**.

---

3. High-Level Design (HLD)
✓ Identifies **major modules** and their interactions.
✓ Defines **data flow** between modules.

Example: In an ATM system, HLD defines how the **user interface**, **database**, and **network modules** interact.

---

4. Low-Level Design (LLD)
✓ Detailed **design of each module**.
✓ Defines **logic, functions, and algorithms** for coding.

Example: In an ATM system, LLD defines how **PIN validation** and **cash withdrawal logic** work.

---

5. Coding (Implementation)
✓ Writing **actual code** for the software.
✓ Follows coding standards and best practices.

Example: Writing **Java/Python** code for the ATM transaction system.

Right Side: Validation (Testing Phases)

Each testing phase corresponds to a development phase to ensure quality.

1. Unit Testing *(Checks individual modules)*
✓ Tests **each function/module** separately.

Example: Test if the ATM correctly deducts money when a withdrawal is made.

---

2. Integration Testing *(Checks module interactions)*
✓ Ensures **different modules work together correctly**.

Example: Verify if the ATM's **card reader**, **PIN authentication**, and **bank database** communicate correctly.

---

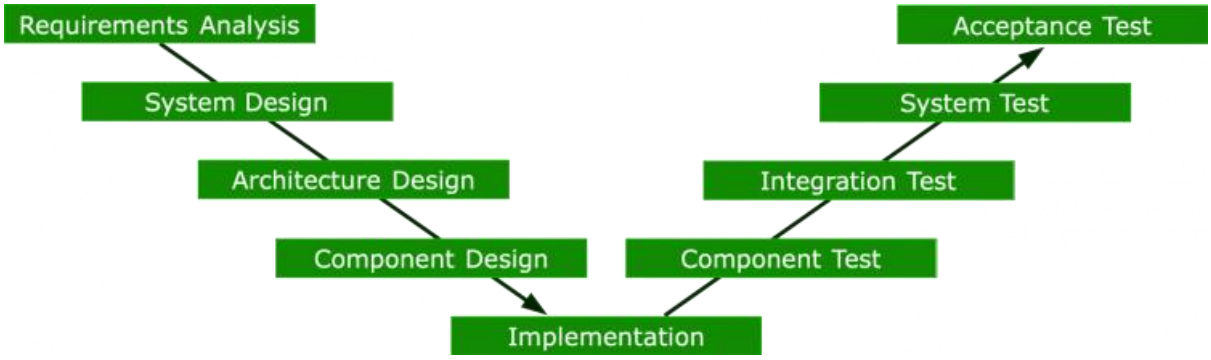
3. System Testing *(Checks full system functionality)*
✓ Tests the entire **software system** under real-world conditions.

Example: Test if the **ATM software** can handle **multiple transactions**, errors, and network failures.

---

4. User Acceptance Testing (UAT) *(Checks customer requirements)*
✓ Performed by **end-users** to validate software meets business needs.

Example: The bank tests the ATM system before launching it for customers.



Advantages of the V-Model

✓ **Early defect detection** → Testing starts in the **requirement phase**, reducing costly errors.
✓ **Better quality assurance** → Each phase is verified before proceeding.
✓ **Simple and easy to use** → Clear structure, making project management easier.
✓ **Works well for critical systems** → Used in **healthcare, banking, and aerospace software** where failure is costly.

Disadvantages of the V-Model

✓ **Rigid and inflexible** → Changes in requirements are **difficult to accommodate**.
✓ **Expensive for large projects** → Detailed documentation and early testing require more effort.
✓ **Late working prototype** → The product is **not available for testing** until late in the development.

When to Use the V-Model?

✓ **For projects with well-defined and stable requirements**.
✓ **For safety-critical applications** (e.g., medical devices, defense systems).
✓ **When quality is the top priority** and rigorous testing is required.

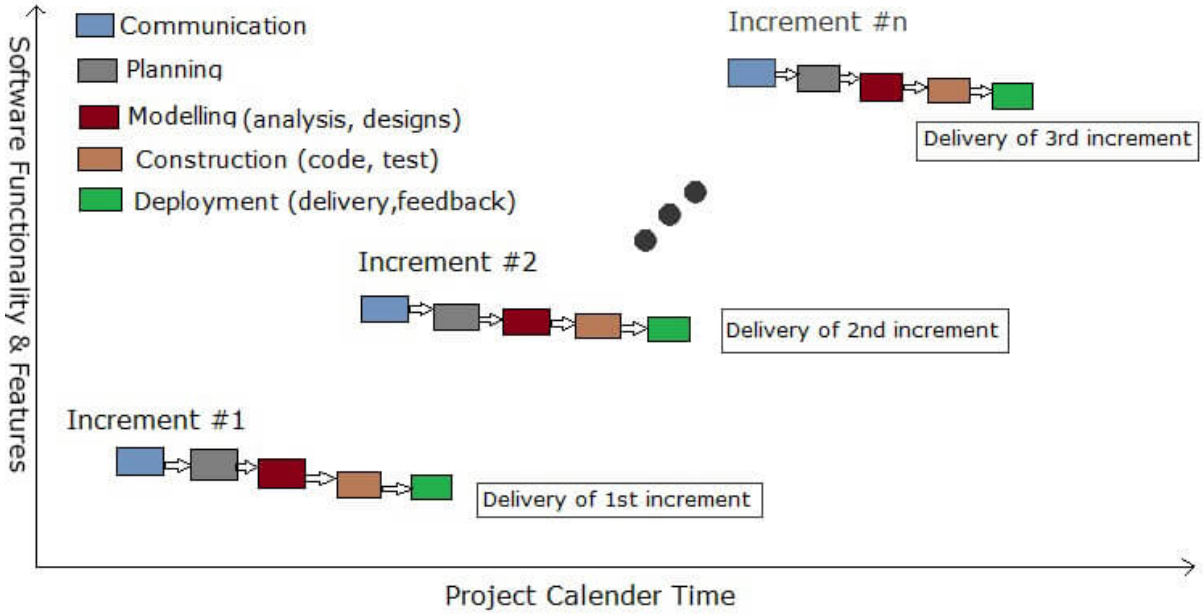
When NOT to Use the V-Model?

✓ **For projects with frequently changing requirements**.
✓ **For Agile development**, where continuous customer feedback is required.
✓ **For startups or fast-changing software**, where flexibility is needed.

Q. Explain Incremental Model

The **Incremental Model** is a **software development process** where the system is built **gradually** in **small parts (increments)**. Each increment **adds functionality** to the previous version until the complete system is developed.

✓ It follows a **"Build a little, test a little"** approach.
✓ It is a combination of **Waterfall and Iterative Models**.



Phases of the Incremental Model

The **Incremental Model** consists of multiple **cycles**, each having the following phases:

1. Requirement Analysis
✓ Identify the **core requirements** first.
✓ Additional requirements are **added in increments**.

Example: In an e-commerce website, the first increment includes **user login** and **product listing**, while later increments add **shopping cart** and **payment system**.

---

2. Design
✓ Design the **basic system architecture** in the first increment.
✓ Each new increment **refines** and **expands** the design.

Example: The first version of an ATM software may have only **cash withdrawal**, while later versions add **balance inquiry** and **fund transfers**.

---

3. Implementation (Coding & Development)

✓ The **first increment** is coded, tested, and released.

✓ Additional increments are developed in **subsequent cycles**.

Example: A mobile banking app may first launch **fund transfer**, then later add **bill payments** and **investment tracking**.

---

4. Testing

✓ Each increment is **tested separately** before being integrated.

✓ Ensures **early error detection** and **continuous validation**.

Example: In an airline reservation system, ticket booking may be tested first, followed by **cancellation and refund processing** in later stages.

---

5. Integration & Deployment

✓ All tested increments are **combined** into the final system.

✓ Ensures **smooth integration** of different components.

Example: A hospital management system may initially support **patient registration**, then later integrate **doctor appointments and billing**.

Advantages of the Incremental Model

✓ **Early Software Delivery** → Users get a **working version** in the first increment.

✓ **Flexibility** → Changes can be made in later increments.

✓ **Lower Risk** → Bugs are fixed in **small phases** instead of the entire system at once.

✓ **Better User Feedback** → Users can test **early versions** and suggest improvements.

✓ **Easy to Manage** → Each phase is small and simple, making management easier.

Disadvantages of the Incremental Model

✓ **Requires Good Planning** → Early decisions affect later increments.

✓ **System Architecture Challenges** → If not planned well, adding new increments can become complex.

✓ **More Testing Effort** → Each increment requires **separate testing**, increasing workload.

✓ **Not Suitable for Rapid Changes** → If requirements change frequently, Agile models like **Scrum** are better.

When to Use the Incremental Model?

✓ **For projects with well-defined core requirements** but additional features evolve over time.
✓ **For large software projects** where dividing into smaller parts helps manage complexity.
✓ **For software requiring early releases**, like web applications, banking software, and e-commerce sites.

When NOT to Use the Incremental Model?

✓ **For small projects**, where building the entire system at once is easier.
✓ **For projects with rapidly changing requirements**, where Agile models (Scrum, Kanban) work better.

Q. What is Evolutionary Model

These models allow software to evolve over time, making them more flexible and adaptable.

a) Prototyping Model

A prototype (working model) is built first, refined based on user feedback, then finalized.

Advantages:

- Reduces misunderstandings about requirements.
- Helps in gathering customer feedback early.

Disadvantages:

- Can lead to scope creep (constant changes delay project completion).

Example: Used in UI/UX design and customer-facing apps.

b) Spiral Model

Combines elements of Waterfall and Prototyping with a strong focus on risk management. The project goes through multiple iterations (spirals), each involving:

- **Planning → Risk Analysis → Development → Evaluation**

Advantages:

- Best for high-risk and complex projects.
- Allows continuous refinement of the product.

Disadvantages:

- Expensive and time-consuming.
- Requires expert risk management.

Example: Used in defence and aerospace projects where risk management is crucial.

Q. Explain Prototyping Model

The **Prototyping Model** is a **software development approach** where a **prototype (working model)** is built first and improved based on **customer feedback** before developing the final system.

✓ It is **iterative**, meaning multiple versions of the prototype are created.
✓ It helps in **understanding unclear requirements** and refining the final product.

Phases of the Prototyping Model

1. Requirement Gathering & Analysis

✓ Collect **initial** requirements from the customer.
✓ Requirements **don't need to be complete** at this stage.

Example: A client wants a **food delivery app** but is unsure about the exact features.

---

2. Quick Design (Initial Prototype)

✓ A **basic UI/UX design** is created.
✓ Focuses on **user interaction** and **basic functionality** (no complex coding).

Example: The food delivery app **prototype** may show restaurant listings, but **ordering & payments won't work yet**.

---

3. Prototype Development

✓ A **working model** is developed with **basic functionality**.
✓ The prototype may be **incomplete or simplified**.

Example: The app **lets users select food items** but doesn't process payments yet.

---

4. User Evaluation & Feedback

✓ The prototype is **shared with the customer** for review.
✓ Customer **suggests changes** and **adds missing features**.

Example: Users might request **filters for vegetarian food** or **multiple payment options**.

---

5. Refining the Prototype (Iterative Improvement)

✓ Based on **feedback**, the prototype is **modified & improved**.
✓ This process repeats until the customer is **satisfied**.

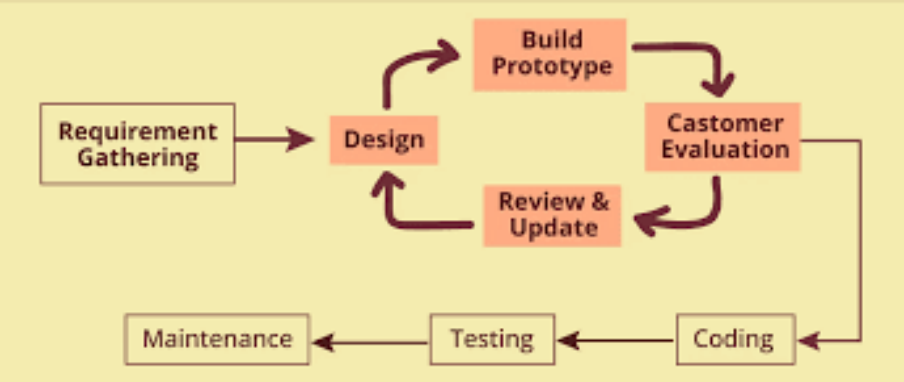
Example: The app is updated to **include payment options and order tracking**.

---

6. Final Product Development & Deployment

✓ Once the prototype is **finalized**, full development begins.
✓ The **final system is tested and deployed**.

Example: The fully developed food delivery app is **released to the public**.



Advantages of the Prototyping Model

✓ **Reduces Misunderstandings** → Customers see a working model before full development.
✓ **User Involvement** → Users provide **continuous feedback**, improving satisfaction.
✓ **Early Bug Detection** → Issues are fixed in the prototype stage, reducing future costs.
✓ **Better Requirement Clarity** → Works well when requirements are **unclear or evolving**.
✓ **Faster Development** → Initial prototypes can be built **quickly**.

Disadvantages of the Prototyping Model

✓ **High Cost** → Continuous changes may increase development cost.
✓ **Scope Creep** → Users may keep **adding new features**, delaying the final product.
✓ **Not Suitable for Large Projects** → Iterative updates may become **difficult to manage**.
✓ **Incomplete Prototypes May Confuse Users** → Users may think the prototype is the final product.

When to Use the Prototyping Model?

✓ **For projects with unclear requirements**, where customer feedback is needed.
✓ **For UI/UX-heavy applications**, like **mobile apps and web apps**.
✓ **For R&D projects**, where multiple revisions are expected.

When NOT to Use the Prototyping Model?

✓ **For small, simple projects** where requirements are well-defined.
✓ **For projects with strict timelines**, as constant revisions can delay the final product.
✓ **For large enterprise systems**, where detailed upfront planning is required.

Q. What are the potential problems of prototyping models

The **Prototyping Model** is useful for gathering user feedback and refining requirements, but it comes with several **challenges and drawbacks**.

---

1. Scope Creep (Uncontrolled Changes in Requirements)

- Users **keep requesting changes** since they see the prototype evolving, leading to **constant modifications**.
- This increases **development time and cost**, making it hard to finalize the project.

2. Increased Development Cost & Time

- Multiple iterations of prototyping **consume more resources** (time, effort, money).
- If not managed properly, the project can **exceed budget and deadlines**.

3. Incomplete or Misleading Prototype

- A prototype may **lack full functionality**, giving users **false expectations**.
- Users may assume that the prototype is **the final system**, leading to **disappointment** if the actual product looks different.

4. Poor Documentation

- Since the focus is on rapid changes, developers may **skip proper documentation**.
- Lack of documentation makes it **hard to maintain and scale** the software later.

5. Developer Dependency on User Feedback

- Developers **rely too much on user input**, which may be **unclear or inconsistent**.
- If users **cannot articulate their needs clearly**, the prototype may not represent the correct requirements.

6. Technical Debt & Poor Architecture

- Rapid changes lead to **quick fixes instead of well-planned solutions**.
- The final system may **lack proper structure**, making it **difficult to maintain and enhance**.

7. Not Suitable for Large-Scale Systems

- Prototyping is ideal for **small to medium projects** but **not for complex systems**.
- Large systems require **detailed planning and structured development**, which prototyping often skips.

Q. Explain Spiral Model

The **Spiral Model** is a **risk-driven, iterative** software development model that combines elements of **Waterfall and Prototyping**. It is mainly used for **large, complex, and high-risk** projects.

✓ **Developed by Barry Boehm in 1986.**
✓ **Best for projects with evolving requirements.**
✓ **Focuses on continuous risk assessment and improvement.**

Phases of the Spiral Model

The **Spiral Model** consists of **four major phases**, which repeat in multiple **spirals (iterations)** until the final product is completed.

1. Planning Phase

✓ Identify **requirements** and objectives for the iteration.
✓ Perform **feasibility study** and estimate cost & timeline.
✓ Identify and analyze **potential risks**.

Example: In an online banking system, the first iteration may focus on **user login and account balance features**.

---

2. Risk Analysis Phase

✓ Identify and analyze **project risks** (technical, financial, operational).
✓ Develop **risk mitigation strategies**.
✓ If a major risk is found, a **prototype is built** to test feasibility.

Example: If online banking security is a risk, a **prototype login system with encryption** is developed first.

---

3. Development & Testing Phase

✓ **Design, code, and test** the selected module.
✓ Uses Agile-like **incremental development**.
✓ Each iteration improves based on testing and feedback.

Example: In an e-commerce app, the **shopping cart feature** is implemented and tested in the first cycle.

---

4. Evaluation & Review Phase

✓ The customer and stakeholders review the progress.
✓ Based on feedback, **changes and improvements** are made.
✓ If needed, another **spiral (iteration)** is started.

Example: After testing the shopping cart, customers request **multiple payment options**, so a new cycle is started.

Advantages of the Spiral Model

✓ **Early Risk Management** → Risks are identified and resolved early.
✓ **Flexibility** → Can adapt to changing requirements.
✓ **Customer Feedback** → Each iteration involves review & feedback.
✓ **Better Cost Estimation** → Costs are estimated in each spiral.
✓ **Suitable for Large Projects** → Works well for complex systems.

Disadvantages of the Spiral Model

✓ **Expensive** → Requires high cost and effort for risk analysis.
✓ **Time-Consuming** → Multiple iterations can delay project completion.
✓ **Requires Skilled Experts** → Risk assessment needs experienced developers.
✓ **Difficult to Manage** → Complex for small projects.

When to Use the Spiral Model?

✓ **For large, complex, and high-risk projects**.
✓ **For software with frequently changing requirements**.
✓ **For mission-critical applications** like banking, defense, and aerospace.

When NOT to Use the Spiral Model?

✓ **For small projects** where cost and risk are low.
✓ **For projects with fixed requirements** (Waterfall is better).
✓ **When quick delivery is needed** (Agile is better).

Q. Differentiate between

Feature	Waterfall Model	V-Model	Incremental Model	Prototyping Model	Spiral Model
Approach	Linear & Sequential	Linear with Testing at Each Stage	Iterative & Incremental	Iterative with Rapid Prototyping	Iterative with Risk Management
Flexibility	Rigid (No Changes Allowed)	Rigid (No Changes Allowed)	Moderate (Allows Changes in Later Increments)	High (Frequent Changes Allowed)	Very High (Changes Allowed in Every Iteration)
Risk Handling	Weak (Risk Identified Late)	Weak (Risk Identified Late)	Medium (Each Increment is Tested)	High (Early Risk Identification)	Very High (Continuous Risk Analysis)
Customer Involvement	Low (Only at Start & End)	Low (Only at Start & End)	Medium (Feedback After Each Increment)	High (Continuous Customer Feedback)	High (User Reviews in Each Spiral)
Testing Phase	Done After Development	Done at Each Phase	Testing After Each Increment	Testing in Prototypes	Testing in Each Spiral
Cost of Errors	High (Errors Found Late)	High (Errors Found Late)	Medium (Errors Fixed in Phases)	Low (Errors Found Early in Prototypes)	Low (Errors Managed in Risk Analysis)
Time to Market	Slow (Final Product at End)	Slow (Final Product at End)	Faster (First Version Available Early)	Fast (Basic Prototype Available Quickly)	Medium (First Working Version Takes Time)
Best For	Small, Well-Defined Projects	Safety-Critical Projects	Medium to Large Projects	Projects with Changing Requirements	Large, High-Risk Projects

Feature	Waterfall Model	V-Model	Incremental Model	Prototyping Model	Spiral Model
Not Suitable For	Dynamic Projects	Dynamic Projects	Small, Simple Projects	Projects with Fixed Requirements	Small, Low-Risk Projects

Q. Explain Component-Based Development Model, Aspect Oriented Development Model, Formal Methods Model

1. Component-Based Development (CBD) Model

The **Component-Based Development (CBD) Model** focuses on building software using **pre-existing, reusable components** instead of developing everything from scratch.

Key Features:

✓ Software is assembled using **independent, modular components**.
✓ Each component has **well-defined interfaces** and can be reused in different applications.
✓ Reduces **development time and cost**.
✓ Improves **software maintainability and scalability**.

Example:

- A **login system** used in multiple websites.
- **E-commerce websites** using a ready-made **payment gateway API (PayPal, Razorpay, Stripe).**

Advantages:

✓ Faster development due to reusability.
✓ Reduces coding effort and cost.
✓ Enhances **scalability** and **maintainability**.

Disadvantages:

✓ Integrating third-party components can be **challenging**.
✓ Customization is **limited** compared to fully custom-built software.

---

2. Aspect-Oriented Development (AOD) Model

The **Aspect-Oriented Development (AOD) Model** is an advanced software design technique that separates **cross-cutting concerns** (features that affect multiple modules, like security and logging).

Key Features:

✓ Improves **modularity** by separating **core business logic** from additional functionalities.
✓ Uses **"Aspects"** to handle cross-cutting concerns **without modifying main code**.
✓ Works alongside **Object-Oriented Programming (OOP)**.

Example:

- **Logging system** that records every user action without modifying the core application.
- **Security modules** that apply authentication rules across different parts of an application.

Advantages:

✓ Increases **modularity** and **code maintainability**.
✓ Reduces **code duplication**.
✓ Enhances **scalability**.

Disadvantages:

✓ Difficult to implement for **small projects**.
✓ Requires **specialized tools** (like AspectJ for Java).

---

3. Formal Methods Model

The **Formal Methods Model** is a **mathematically-based approach** used for **developing highly reliable and secure software systems**.

Key Features:

✓ Uses **mathematical models, logic, and proofs** to verify software correctness.
✓ Eliminates software **errors before coding starts**.
✓ Commonly used in **critical systems** (aerospace, military, medical devices).

Example:

- **Air traffic control systems** using formal verification to prevent failures.
- **NASA software** for space missions, ensuring zero errors.

Advantages:

✓ Ensures **high reliability** and **error-free** software.
✓ Detects **design flaws early**.
✓ Essential for **safety-critical applications**.

Disadvantages:

✓ **Complex and costly** to implement.
✓ Requires **specialized mathematical knowledge**.
✓ Not practical for **small, non-critical applications**.

Q. Explain umbrella activities of software engineering

In software engineering, **umbrella activities** are **supportive tasks** that span across all phases of the **software development life cycle (SDLC)**. They ensure **quality, consistency, and control** over the development process.

These activities **do not belong to a single phase** but instead work alongside all phases to ensure a successful software project.

Key Umbrella Activities:

1. Software Project Tracking & Control

- Ensures the project stays on **schedule, within budget, and meets quality standards**.
- Uses project management techniques like **Gantt charts, Earned Value Analysis (EVA), and status meetings**.

2. Risk Management

- Identifies, analyzes, and mitigates **potential risks** in the project.
- Risks include **technical issues, budget overruns, and schedule delays**.
- Example: Using **Risk Mitigation, Monitoring, and Management (RMMM)** strategies.

3. Software Quality Assurance (SQA)

- Ensures that the software **meets quality standards**.
- Includes **reviews, audits, testing, and defect tracking**.

4. Software Configuration Management (SCM)

- Manages **changes** to software (code, documentation, requirements, etc.).
- Uses **version control systems** like Git to track modifications.

5. Software Reviews

- Conducts **peer reviews, walkthroughs, and inspections**.
- Helps to **detect errors early** before testing.

6. Software Reusability Management

- Promotes the **use of existing components** to **save time and cost**.
- Example: Using **pre-built libraries or modules** instead of coding from scratch.

7. Measurement & Metrics

- Collects data on **software performance, defects, and productivity**.
- Example: Using **Lines of Code (LOC), Function Points (FP), and Cyclomatic Complexity** to measure software size and complexity.

8. Document Management

- Maintains **technical documents, reports, and manuals** throughout the project.
- Includes **Software Requirement Specification (SRS), Design Documents, and User Manuals**.

Importance of Umbrella Activities:

✓ **Ensures software quality**
✓ **Helps in risk mitigation**
✓ **Improves team coordination**
✓ **Reduces cost & time wastage**
✓ **Maintains consistency in the project**

Q. Write short note on: Security Engineering

Security Engineering is the field of **designing, building, and maintaining secure systems** to protect software, hardware, networks, and data from threats, attacks, and vulnerabilities. It integrates principles from **computer science, cryptography, network security, and risk management** to ensure **confidentiality, integrity, and availability (CIA) of information**.

Goals of Security Engineering (CIA Triad)

1. Confidentiality → Protect sensitive data from unauthorized access.
2. Integrity → Ensure data is accurate and cannot be modified without authorization.
3. Availability → Ensure systems and data are accessible when needed.

Key Concepts in Security Engineering

Concept	Description
Threats	Potential dangers that can exploit system vulnerabilities (e.g., hacking, phishing, malware).
Vulnerabilities	Weak points in a system that attackers can exploit (e.g., weak passwords, unpatched software).
Attack Surface	The sum of all possible entry points for attackers (e.g., web applications, APIs, databases).

Concept	Description
Risk Assessment	Identifying and evaluating security risks to mitigate threats.
Security Controls	Mechanisms used to protect systems (e.g., firewalls, encryption, access control).

Phases of Security Engineering

1. Risk Assessment & Threat Modeling

✓ Identify assets (data, software, hardware).
✓ Analyze potential threats and vulnerabilities.
✓ Develop countermeasures to mitigate risks.

2. Secure Design & Development

✓ Use **Secure Software Development Life Cycle (SDLC)** principles.
✓ Implement **secure coding practices** to prevent vulnerabilities like SQL injection and cross-site scripting (XSS).
✓ Use **threat modeling techniques** (e.g., STRIDE, DREAD).

3. Implementation of Security Controls

✓ **Authentication & Authorization** → Use **multi-factor authentication (MFA), role-based access control (RBAC)**.
✓ **Encryption** → Use strong encryption algorithms (**AES, RSA, SSL/TLS**) to protect data.
✓ **Firewalls & Intrusion Detection Systems (IDS/IPS)** → Monitor and block malicious activities.

4. Security Testing

✓ **Penetration Testing** → Simulate attacks to find vulnerabilities.
✓ **Vulnerability Scanning** → Use tools like **Nessus, OpenVAS** to detect security flaws.
✓ **Code Review & Static Analysis** → Identify security weaknesses in code using tools like **SonarQube, Fortify**.

5. Incident Response & Recovery

✓ Monitor logs for suspicious activity.
✓ Have a **disaster recovery plan (DRP)** for handling breaches.
✓ Ensure **regular backups** of critical data.

Security Engineering Techniques

Technique	Purpose
Cryptography	Encrypts sensitive data to prevent unauthorized access.
Access Control	Ensures only authorized users can access specific resources.
Secure Coding	Follows best practices to avoid security vulnerabilities.
Network Security	Uses firewalls, VPNs, and IDS/IPS to protect networks.
Incident Response	Detects and responds to security breaches effectively.

Common Security Threats & Attacks

Threat	Description	Mitigation
Phishing	Deceptive emails trick users into revealing credentials.	Security awareness training, email filtering.
Malware	Viruses, worms, ransomware that harm systems.	Antivirus software, regular updates.
SQL Injection	Attackers manipulate database queries to access data.	Input validation, parameterized queries.
DDoS (Distributed Denial of Service)	Overwhelms a system with traffic to disrupt services.	Firewalls, traffic monitoring, rate limiting.
Zero-Day Exploits	Attackers exploit unknown vulnerabilities before a patch is available.	Regular security updates, intrusion detection.

Importance of Security Engineering

✓ Prevents financial and data loss.
✓ Ensures compliance with regulations (e.g., GDPR, HIPAA, ISO 27001).
✓ Builds trust with customers and users.
✓ Protects intellectual property and business reputation.

Q.