Software Engineering (303105253)

1. Define Software Engineering. How does it differ from traditional programming?
2. What are the major challenges in Software Engineering?
3. Explain the generic view of Software Engineering.
4. List and explain the characteristics of good software.
5. What are the major components of software?
6. Give examples of different types of software applications.
7. Explain the importance of layered technology in software engineering.
8. Define software process, methods, and tools with examples.
9. Why is Software Engineering considered a layered technology?
10. Write a short note on software myths and their impact.
11. Explain the Waterfall Model with its advantages and disadvantages.
12. When is the Waterfall model best suited? Give an example scenario.
13. Differentiate between Incremental and Waterfall models.
14. What is the Incremental Process Model? Mention its advantages.
15. Explain Prototype Model with its phases.
16. What are the major problems solved by the Prototyping approach?
17. Discuss the Spiral Model and its risk analysis concept.
18. Compare Spiral Model vs Incremental Model.
19. What is the Concurrent Development Model? Give an example.
20. Mention two advantages and disadvantages of Evolutionary models
21. Define Software Architecturul design. Why is it important in software design?
22. What are the main principles of Agile Manifesto?
23. Explain Extreme Programming (XP) practices.
24. What is the difference between Agile Model and Waterfall Model?
25. Explain the Scrum framework in Agile development.
26. What are Agile tools? Give examples.
27. How does Agile handle changing requirements during development?
28. Compare Agile vs Spiral model.
29. What is Pair Programming in Agile?
30. What is the Management Spectrum in Software Engineering?
31. Explain the 4 P's principal People-Product-Process-Project relationship or Discuss the **4 P's** of software project management in detail with examples
32. Explain the Product perspective in project management.
31. Define Process in the management spectrum.
32. What is a Software Project? Give examples.
33. What is the purpose of project planning?
34. Define Scope and Feasibility Study in software project planning.
35. Explain Effort Estimation techniques with an example.
36. What is Function Point Analysis (FPA)?
37. What is the COCOMO Model for effort estimation?
38. Define schedule and staffing in a project.
39. Why is team management important in software projects?
40. What is Risk Management in software projects?
41. Explain Risk Identification and Assessment with examples.
42. What is Risk Control?
43. Define Project Monitoring Plan.
44. What is Detailed Scheduling and why is it important?
45. What is the W5HH principle? Explain7 questions of W5HH principle
46. What are the key design concepts in software engineering? Explain abstraction and modularity with example.

Software Engineering (303105253)

47. What is Requirements Specification?
48. What is the importance of Use Cases in requirements?
49. What is a Design Model**?** List and explain its major elements.
50. Differentiate between Functional and Non-functional requirements.
51. What is Requirements Validation?
52. Explain Requirements Analysis in detail.
53. Why is Requirements Gathering important?
54. What are the challenges in Requirements Elicitation?
55. Define Stakeholder Analysis in requirement engineering.
56. What is Feasibility Analysis in requirements engineering?
57. Explain Requirements Management and its benefits.
58. What is the role of prototyping in requirements validation?
59. Differentiate between User Requirements and System Requirements.
60. What are ambiguities in requirements? How to avoid them?
61. Give a real-life example of a Use Case diagram.
62. Explain Process Layer in software engineering layered technology.
63. What is the Method Layer? Give an example.
64. Explain the Tool Layer with a practical example.
65. What is the importance of CASE tools?
66. What are the key design concepts in software engineering? Explain abstraction and modularity with example.
67. What is the relationship between process, methods, and tools?
68. Explain data centered architecture.
69. What are the different categories of software tools?
70. What is Software Quality Assurance (SQA) tools' role?
71. Explain why software projects often fail despite good planning.
72. What are the key differences between Predictive and Adaptive models?
73. What are the ethical issues in software engineering?
74. Explain how risk management improves project success.
75. Give a case study of Waterfall model failure.
76. Discuss advantages of evolutionary models over traditional models.
77. What is the impact of team management on software quality?
78. What are measurable project metrics in software engineering?
79. Give an example of a Risk Matrix in software development.
80. What is the role of a project manager in software projects?
81. What is the importance of documentation in software projects.
82. Explain the concepts of *classes, objects, inheritance, and polymorphism* in OOD.
83. What is an *Entity-Relationship (E-R) diagram*? Explain different types of *attributes* in E-R diagrams and draw an E-R diagram for library management.
84. Define project scope.
85. What is the importance of defining scope in a software project?
86. Differentiate between scope and objectives of a project.
87. What are the main components of a feasibility study?
88. Explain the relationship between scope definition and project cost estimation.
89. What is effort estimation in software project management?
90. Why is effort estimation important?
91. List three popular effort estimation techniques.
92. What is the **COCOMO model**?
93. Differentiate between **COCOMO Basic, Intermediate, and Detailed** models.
94. What is project scheduling?

Software Engineering (303105253)

95. Why is scheduling important in software projects?
96. Define staffing in the context of software projects.
97. Explain the relationship between **effort, schedule, and staffing**.
98. What is risk in software project management?
99. Define **risk identification**. Give examples of **technical**, **business**, and **schedule** risks
100. What is risk assessment? Explain **risk probability and impact matrix**
101. Explain **risk mitigation**, **risk monitoring**, and **risk contingency planning**.
102. What are the common risk mitigation strategies?
103. Differentiate between **risk avoidance** and **risk acceptance**.
104. What is **Earned Value Analysis (EVA)** in project monitoring?
105. Explain the role of **elicitation, analysis, specification, validation, and management** in RE.
106. Define Termonology and its types.
107. What is a Software Requirement Specification (SRS)?
108. What is a use case in software engineering?
109. Explain the structure of a use case (Actor, Precondition, Steps, Postcondition).
110. Differentiate between **primary actor** and **secondary actor** in a use case.
111. What is the difference between a **use case diagram** and a **use case description**?
112. Give an example of a use case for an ATM system.
113. Explain the role of **stakeholders** in validation.
114. Differentiate between **verification** and **validation** of requirements.
115. Differentiate between requirement analysis and requirement elicitation.
116. What is the purpose of an RMMM plan in software project management?
117. What are software metrics? Why are they important? Differentiate between **process metrics, project metrics, and product metrics** with examples.

1. Define Software Engineering. How does it differ from traditional programming?

Ans. Software Engineering is the systematic, disciplined, and quantifiable approach to the design, development, testing, deployment, and maintenance of software.

Difference from Traditional Programming (Short):

- **Scope:** Software engineering focuses on the entire software lifecycle; traditional programming mainly focuses on writing code.
- **Process:** It follows structured processes/models; programming is often ad-hoc.
- **Quality:** Emphasizes scalability, reliability, maintainability; programming may ignore long-term quality.
- **Teamwork:** Software engineering involves teamwork and project management; programming can be individual.

Software Engineering (303105253)

2. What are the major challenges in Software Engineering?

Ans. Major challenges in Software Engineering (short):

- **Managing complexity** of large systems
- **Changing requirements** during development
- **Ensuring quality** (reliability, security, performance)
- **Meeting deadlines** and budgets
- **Team coordination** and communication
- **Maintenance & scalability** after deployment

3. Explain the generic view of Software Engineering.

Ans. The **generic view of software engineering** describes software development as a structured process with key activities:

- **Communication** – Understanding requirements from stakeholders.
- **Planning** – Estimating effort, cost, and scheduling tasks.
- **Modeling** – Designing the software architecture and components.
- **Construction** – Coding and testing the software.
- **Deployment** – Delivering the software and getting feedback.

These activities ensure **systematic, quality-oriented software development.**

4. List and explain the characteristics of good software.

Characteristics of Good Software:

1. **Correctness** – It meets all specified requirements.
2. **Reliability** – Performs consistently without failure.
3. **Efficiency** – Uses resources like CPU, memory optimally.
4. **Maintainability** – Easy to modify, update, and fix.
5. **Usability** – Easy for users to learn and operate.
6. **Portability** – Works on different platforms/environments.
7. **Reusability** – Can be reused in other projects.
8. **Integrity/Security** – Protects data and resists attacks.

5. What are the major components of software?

Major components of software:

1. **Programs (Code)**
   - These are sets of instructions written in a programming language that perform specific tasks or solve problems.
   - Example: The source code of a text editor or calculator.
2. **Documentation**
   - Written descriptions that explain how the software works and how to use it.
   - Includes user manuals, developer guides, and API references.
   - Helps in **maintenance, training, and future updates**.
3. **Data**
   - The information that the software processes, stores, or manipulates.
   - Example: Database records, configuration files, user input/output data.
4. **Software Configuration Items (Supporting Files)**
   - These are additional elements required for the software to function correctly, such as libraries, frameworks, configuration settings, and dependencies.
   - Example: DLL files, JSON/YAML config files, shared libraries.

So, **software = program(s) + documentation + data + configuration/support files**, all working together to provide functionality.

6. Give examples of different types of software applications.

Examples of Different Types of Software Applications:

1. **System Software** – Manages hardware and system resources
   - *Examples:* Windows, Linux, macOS
2. **Application Software** – Helps users perform specific tasks
   - *Examples:* MS Word, Photoshop, VLC Media Player
3. **Utility Software** – Performs maintenance tasks
   - *Examples:* Antivirus, Disk Cleanup, WinRAR
4. **Embedded Software** – Runs on hardware devices
   - *Examples:* Software in washing machines, printers, smart TVs
5. **Web Applications** – Run on web browsers
   - *Examples:* Gmail, Google Docs, Facebook
6. **Mobile Applications** – Run on smartphones
   - *Examples:* WhatsApp, Instagram, Uber

7. Explain the importance of layered technology in software engineering.

Importance of Layered Technology in Software Engineering

Layered technology provides a **structured approach** to develop high-quality software. Its importance is:

1. **Quality Assurance**

- The **quality focus layer** ensures reliability, maintainability, and security in all development activities.

2. **Organized Development Process**
   - The **process layer** defines clear steps (planning, modeling, construction, deployment), making development systematic and manageable.
3. **Technical Guidance**
   - The **methods layer** provides proven techniques for analysis, design, coding, and testing, reducing errors.
4. **Automation and Efficiency**
   - The **tools layer** supports methods and processes with automation (e.g., CASE tools), improving productivity and consistency.
5. **Better Communication**
   - Clear layers help teams understand roles and interactions, improving coordination and reducing complexity.

8. Define software process, methods, and tools with examples.

Ans. Software Process

- **Definition:**
  A **set of activities and tasks** required to develop and maintain software.
- **Purpose:**
  Provides a **framework** for planning, developing, testing, and delivering software.
- **Examples:**
  - Requirements gathering
  - Design, coding, testing, deployment
  - **Process models** like Waterfall, Agile, Spiral

2. Software Methods

- **Definition:**
  The **technical "how-to" approaches** used to build software within the process.
- **Purpose:**
  Helps in **analysis, design, coding, and testing** in a structured way.
- **Examples:**
  - UML diagrams for design
  - ER diagrams for database modeling
  - Structured programming techniques
  - Test case design methods

3. Software Tools

- **Definition:**
  Automated or semi-automated programs that support processes and methods.
- **Purpose:**
  Increase productivity, accuracy, and efficiency.
- **Examples:**
  - **IDE** (Visual Studio, Eclipse) for coding
  - **Version control** (Git)
  - **Testing tools** (Selenium, JUnit)
  - **CASE tools** for design and documentation

- Process = What to do (framework)
- Methods = How to do (techniques)
- Tools = With what to do (automation support)

9. Why is Software Engineering considered a layered technology?

Software Engineering is considered a layered technology because it is organized in **four layers** where each layer supports the one above it:

1. **Quality Focus** – ensures software quality.
2. **Process Layer** – provides the framework for development.
3. **Methods Layer** – gives technical "how-to" for building software.
4. **Tools Layer** – provides automated support for process and methods.

10. Write a short note on software myths and their impact.

Software myths are **false beliefs or wrong assumptions** about software development that many people (managers, customers, or developers) believe to be true. They create **misunderstandings** and lead to poor decision-making.

Types of Software Myths with Examples

1. **Management Myths**
   - *"Adding more people to a late project will make it finish faster."*
   - **Reality:** New members need training, causing more delays.
2. **Customer Myths**
   - *"Software requirements can change easily anytime."*
   - **Reality:** Changes after development increase cost and complexity.
3. **Developer Myths**
   - *"Once software is working, it's easy to modify."*
   - **Reality:** Poorly structured software becomes harder to maintain.

Impact of Software Myths

- Unrealistic **schedules and budgets**

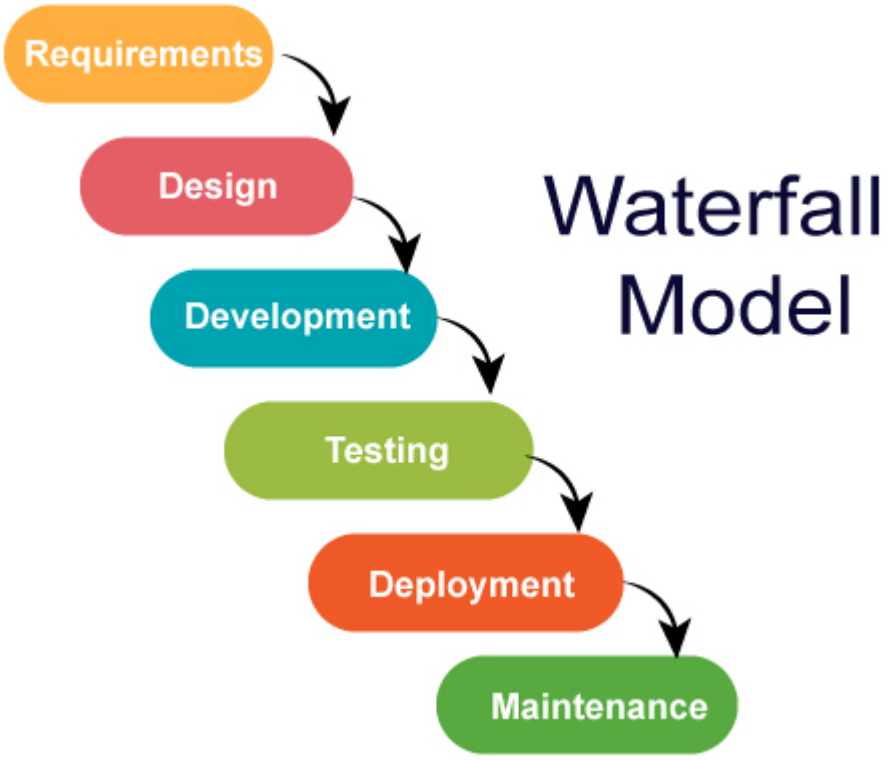
- Poor **resource management**
- Miscommunication between **team and client**
- Increased **project risk, cost, and failure rates**

11. Explain the Waterfall Model with its advantages and disadvantages.

The **Waterfall Model** is a **linear and sequential approach** to software development. In this model, each phase must be **fully completed before moving to the next phase**, like water flowing down a waterfall. It is one of the **earliest SDLC models**.

Phases of the Waterfall Model

1. **Requirement Analysis**
   - Collect and document all user requirements.
   - Output: Software Requirement Specification (SRS).
2. **System Design**
   - Plan the system architecture, database design, UI design.
   - Output: Design documents.
3. **Implementation/Devolopment(Coding)**
   - Developers write code based on the design.
   - Output: Source code/program modules.



4. **Testing**
    - Verify and validate the software against requirements.
    - Output: Tested and bug-fixed software.
5. **Deployment**
    - Deliver the software to users.
6. **Maintenance**
    - Fix issues, update features after delivery.

Advantages of the Waterfall Model

- **Simple and easy** to understand and manage.
- **Clear structure** with well-defined phases.
- Works well for **small projects with fixed requirements**.
- **Documentation** is maintained at every phase.

Disadvantages of the Waterfall Model

- **Rigid:** Hard to go back to a previous phase if changes are needed.
- **Late error detection:** Problems are found only in the testing phase.
- Not suitable for **complex or evolving requirements**.
- **Customer feedback** is received late, after completion.

12. When is the Waterfall model best suited? Give an example scenario.

When is the Waterfall Model best suited?

- It is best for **small projects with well-defined, stable, and clear requirements** where changes are unlikely.
- Suitable when **technology is well understood** and risks are low.

Example Scenario:

- Developing a **banking calculator software** with fixed features like interest calculation, where requirements won't change.

13. Differentiate between Incremental and Waterfall models.

Ans. Difference between Incremental and Waterfall Models

Aspect	Waterfall Model	Incremental Model

Software Engineering (303105253)

Aspect	Waterfall Model	Incremental Model
Process Flow	Linear, sequential phases.	Developed in **small increments** (partial systems).
Flexibility	Rigid, hard to accommodate changes.	More flexible, allows changes after each increment.
Delivery	Entire system delivered at the end.	Working software delivered **in parts** after each increment.
Risk Handling	High risk if requirements change.	Lower risk as feedback is taken in each iteration.
Example	Payroll system with fixed requirements.	E-commerce app developed in modules (login, catalog, payment).

14. What is the Incremental Process Model? Mention its advantages.

Ans. Incremental Process Model:

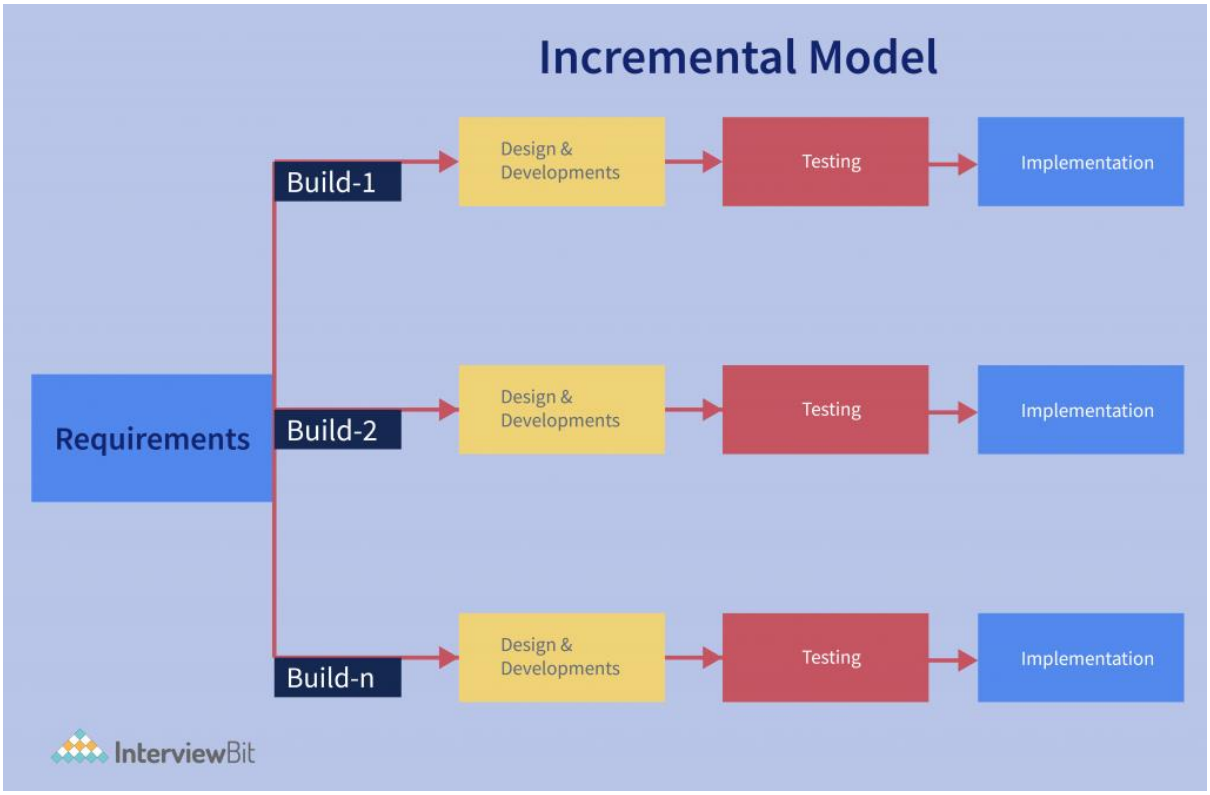
- It is a **software development model** where the system is built **in small increments (modules)**.
- Each increment adds **new features**, and after several increments, the complete system is delivered.

Key Features

- Development is **divided into smaller parts** (increments).
- Each increment results in a **working version of the software**.
- Feedback is collected after each increment to improve the next one.
- Combines elements of **linear (Waterfall)** and **iterative** approaches.

Phases in Each Increment

1. **Requirements analysis**
2. **Design & Devilopments**
3. **Testing & integration**
4. **Implementation**

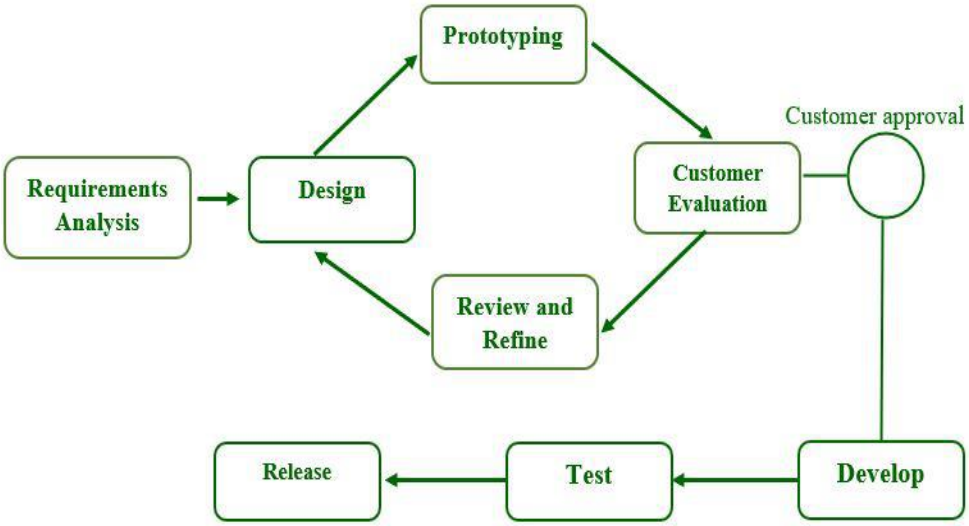


Advantages:

- Early **partial working software** is available.
- Easier to **manage changes** in requirements.
- **Lower risk** as problems are detected early.
- Faster **customer feedback** and improved satisfaction.

15. Explain Prototype Model with its phases.

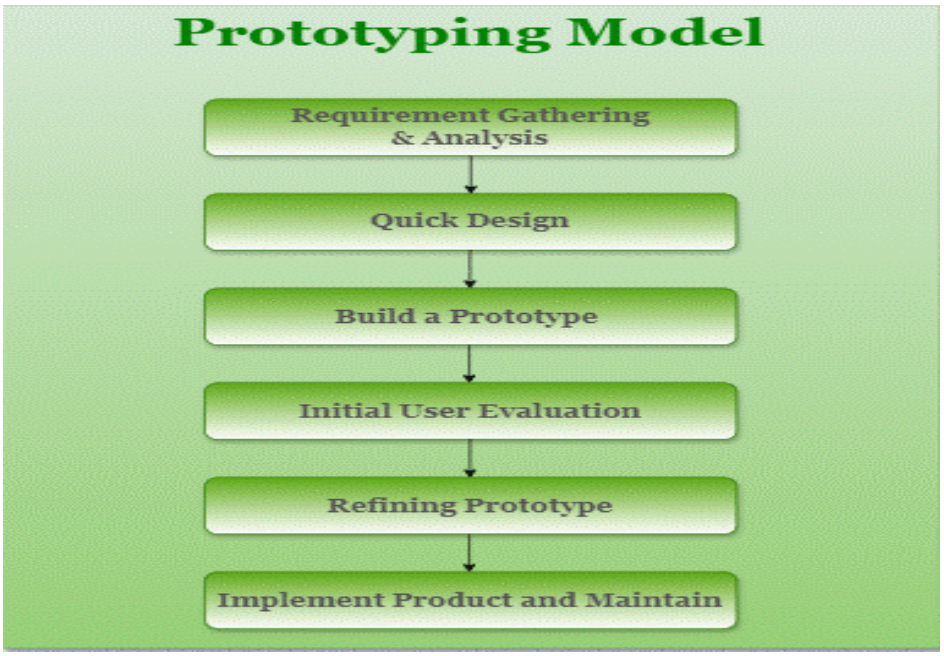
Ans. The Prototyping Model is one of the most popularly used **Software Development Life Cycle Models (SDLC models)**. This model is used when the customers do not know the exact project requirements beforehand.



In this process model, the system is partially implemented before or during the analysis phase thereby allowing the customers to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model.

The **Prototype Model** is a software development approach where a **prototype (working model)** of the system is built early to understand requirements better and get user feedback.

Phases of Prototype Model



1. **Requirement Identification** – Collect initial requirements, focusing on unclear or complex parts.
2. **Quick Design** – Create a basic design for the prototype (UI, main features).
3. **Prototype Building** – Develop a simple, working model.

4. **User Evaluation** – Users test and give feedback on the prototype.
5. **Refinement** – Improve the prototype based on feedback until requirements are clear.
6. **Final System Development(product and maintain)** – Develop the actual system with well-understood requirements.

16. What are the major problems solved by the Prototyping approach?

Ans. The **Prototyping approach** solves major problems like:

- **Unclear or incomplete requirements** → Helps clarify user needs early.
- **Misunderstanding between users and developers** → Improves communication through visual models.
- **High risk of wrong design** → Detects design issues early.
- **Late discovery of errors** → Finds problems before final development.

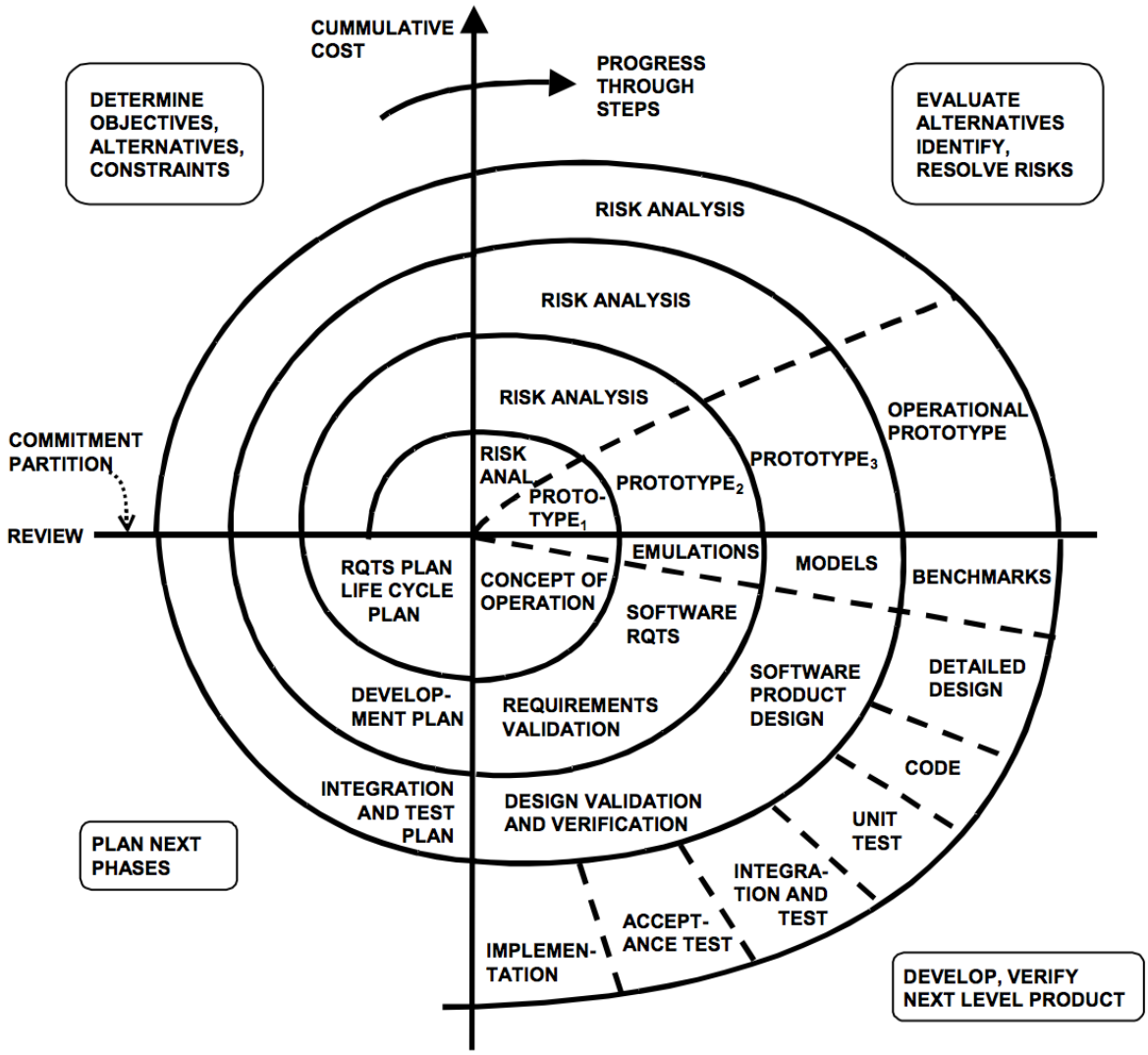
17. Discuss the Spiral Model and its risk analysis concept.

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

It works in **four phases per spiral (iteration):**

1. **Planning** – Identify objectives, alternatives, and constraints.
2. **Risk Analysis** – Identify and analyze risks, and develop strategies to reduce them.
3. **Engineering** – Develop prototypes or actual system components.
4. **Evaluation** – Get user feedback and plan for the next iteration.

This process **repeats in spirals** until the final system is built.



risk Analysis Concept

- **Risk identification** – Find potential risks (technical, cost, schedule).
- **Risk assessment** – Analyze the impact and likelihood of each risk.
- **Risk mitigation** – Take preventive steps (e.g., building prototypes, simulations).

Example: If a new technology is uncertain, a small prototype is built first to reduce risk.

18. **Compare Spiral Model vs Incremental Model**.

Spiral Model vs Incremental Model

Aspect	Spiral Model	Incremental Model
Focus	Risk-driven, emphasizes **risk analysis**	Delivers software in **small increments**
Development	Iterative with **risk evaluation** in	Iterative with **working parts**

Aspect	Spiral Model	Incremental Model
Style	each cycle	**delivered step by step**
Risk Handling	High priority – risks identified & mitigated early	Minimal risk analysis, assumes stable requirements
Complexity	Suitable for **large, complex, high-risk** projects	Suitable for **medium-sized, less complex** projects
Cost	Can be costly due to repeated risk analysis	Relatively cheaper
User Involvement	Continuous user evaluation after each spiral	User feedback after each increment
End Product	Final system evolves after multiple spirals	System grows by adding functional increments

- **Spiral** = Best for high-risk, complex projects.
- **Incremental** = Best for projects needing quick partial deliveries.

19. What is the Concurrent Development Model? Give an example.

The **Concurrent Development Model** is an **evolutionary software process model** where **different activities (like design, coding, testing)** happen **in parallel** rather than sequentially.

- It represents software development as a **network of activities**, each having states like *waiting, under development, completed*.
- When an event occurs (e.g., a requirement change), it triggers related activities dynamically.

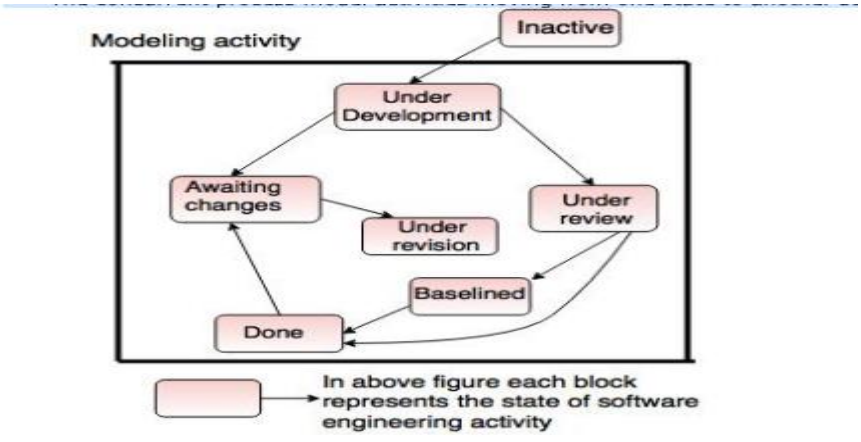


Fig. - One element of the concurrent process model

Key Idea

Unlike Waterfall (linear) or Incremental (stepwise), **Concurrent model allows multiple phases to overlap**, enabling **faster delivery** and **quick handling of changes**.

How it Works

- The model is represented as a **series of events and states** for each activity.
- At any time, different parts of the software may be at different stages.
- Example:
  - While *analysis* is in **under development** state, *design* may still be in **waiting** state.
  - When *design* moves to **under development**, *implementation* may move to **waiting**.

So the model adapts dynamically to changes, unlike the rigid sequential models like Waterfall.

Example

- While **designing a module**, testing activities like *test case preparation* may start in parallel.
- If requirements change, the model immediately updates related design and coding activities.

[Requirement Analysis] ↔ [Design] ↔ [Implementation] ↔ [Testing] ↔ [Maintenance]

↕ ↕ ↕

(States: Waiting → Under Development → Reviewed → Completed)

20. Mention two advantages and disadvantages of Evolutionary models

Advantages of Evolutionary Models

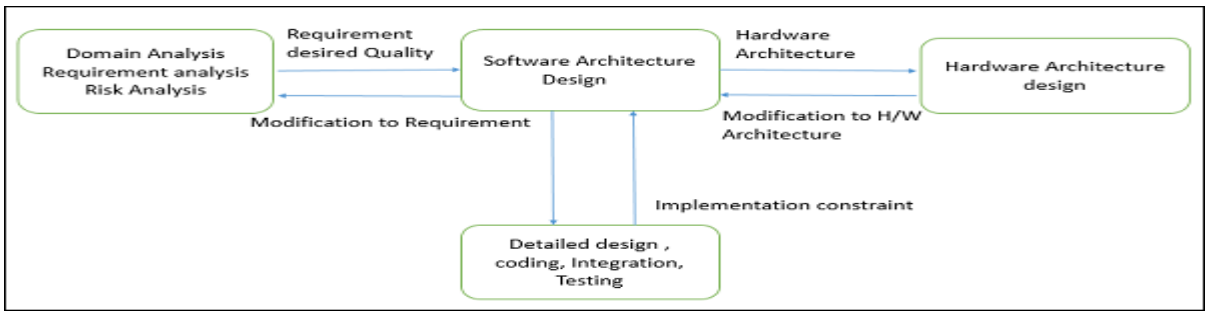
1. **Better requirement understanding** – Users see early versions and give feedback, reducing misunderstandings.
2. **Early delivery of working software** – Useful features are available sooner for testing and use.

Disadvantages of Evolutionary Models

1. **May increase project cost** – Multiple iterations can require more time and resources.
2. **Poor documentation** – Frequent changes may lead to incomplete or inconsistent documentation.

21. Define Software Architecturul design. Why is it important in software design?

Ans. **Software Architectural Design** refers to the high-level structure of a software system, defining its components, their relationships, interactions, and guiding principles for development.



Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows –

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

Importance:

- Provides a clear blueprint for system development.
- Improves scalability, maintainability, and performance.
- Helps manage complexity and supports better decision-making.
- Ensures system quality attributes like security, reliability, and flexibility.

22. What are the main principles of Agile Manifesto?

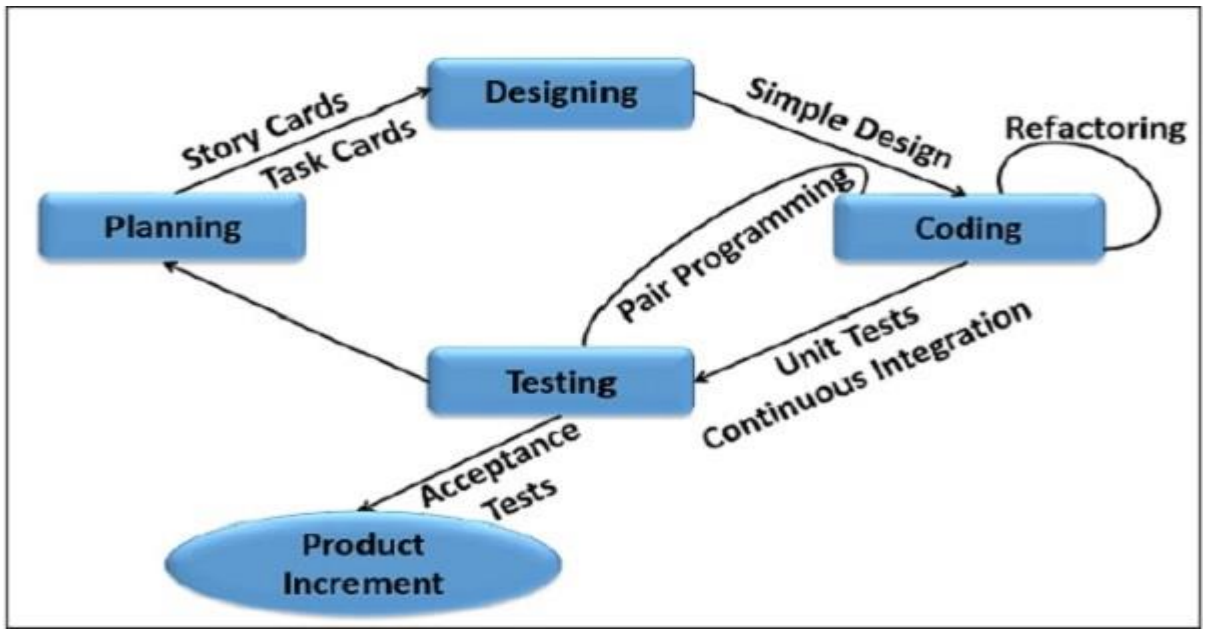
Ans. The main principles of the Agile Manifesto are:

1. **Customer satisfaction** through early and continuous software delivery.
2. **Welcome changing requirements** even late in development.
3. **Frequent delivery** of working software.
4. **Close collaboration** between business people and developers.
5. **Motivated individuals** with trust and support.
6. **Face-to-face communication** is most effective.
7. **Working software** is the primary measure of progress.
8. **Sustainable development** with a constant pace.
9. **Technical excellence** and good design improve agility.
10. **Simplicity**—maximize work not done.

11. **Self-organizing teams** produce the best architectures and designs.
12. **Regular reflection** to improve effectiveness.

23. Explain Extreme Programming (XP) practices.

Extreme Programming (XP) is an Agile software development methodology focused on improving software quality and responsiveness to changing requirements through frequent releases in short development cycles.

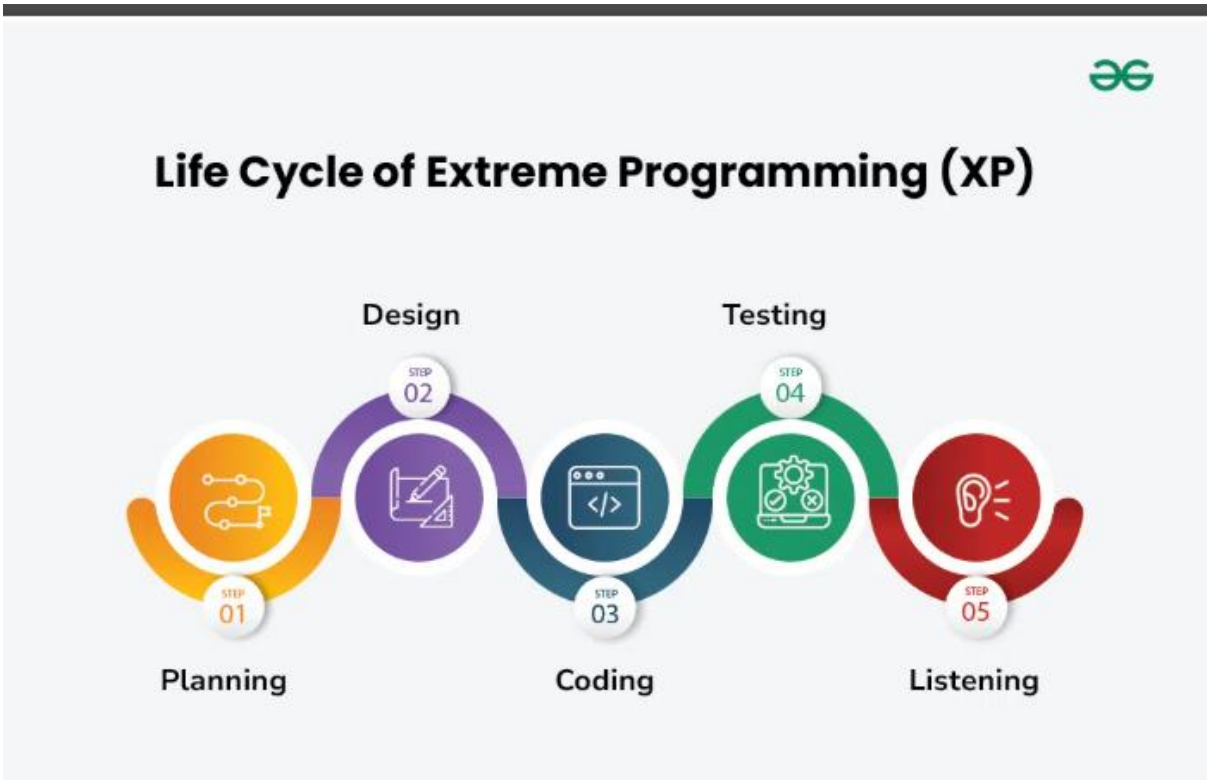


This diagram represents **Extreme Programming (XP)**, an **Agile development methodology**.

- **Planning** – Uses *story cards* and *task cards* to plan features.
- **Designing** – Focuses on *simple design* that can evolve.
- **Coding** – Done with *pair programming* and continuous *refactoring* to improve code.
- **Testing** – Includes *unit tests*, *continuous integration*, and *acceptance tests*.
- After these steps, you get a **Product Increment**, a working improved version of the software.

Life Cycle of Extreme Programming (XP)

The Extreme Programming Life Cycle consist of five phases:



1. **Planning:** The first stage of Extreme Programming is planning. During this phase, clients define their needs in concise descriptions known as user stories.
2. **Design:** The team creates only the essential design needed for current user stories, using a common analogy or story to help everyone understand the overall system architecture and keep the design straightforward and clear.
3. **Coding:** Extreme Programming (XP) promotes pair programming i.e**.** wo developers work together at one workstation, enhancing code quality and knowledge sharing. They write tests before coding to ensure functionality from the start (TDD).
4. **Testing:** Extreme Programming (XP) gives more importance to testing that consist of both unit tests and acceptance test**.** Unit tests, which are automated, check if specific features work correctly.
5. **Listening:** In the listening phase regular feedback from customers to ensure the product meets their needs and to adapt to any changes.

Key XP Practices:

1. **Pair Programming** – Two developers work together on the same code, improving code quality and knowledge sharing.
2. **Test-Driven Development (TDD)** – Writing tests before coding ensures functionality meets requirements.
3. **Continuous Integration** – Code is integrated and tested frequently to detect issues early.
4. **Small Releases** – Frequent, incremental releases provide quick feedback.
5. **Simple Design** – Keep the design as simple as possible, avoiding unnecessary complexity.
6. **Refactoring** – Continuously improving existing code without changing its functionality.

7. **Collective Code Ownership** – Any developer can improve any part of the codebase.
8. **Coding Standards** – Maintain consistent coding practices for readability.
9. **Sustainable Pace** – Work at a pace that can be maintained indefinitely (no burnout).
10. **Customer Involvement** – On-site customer for continuous feedback and clarification.

Advantages of Extreme Programming (XP)

- **Slipped schedules**
- **Misunderstanding the business and/or domain**
- **Canceled projects**
- **Staff turnover**
- **Costs incurred in changes**
- **Business changes**
- **Production and post-delivery defects**

24. What is the difference between Agile Model and Waterfall Model?

Ans. Here's a clear **comparison between Agile Model and Waterfall Model**:

Aspect	Waterfall Model	Agile Model
Approach	Linear and sequential	Iterative and incremental
Flexibility	Rigid, changes are difficult	Flexible, changes are easily adapted
Customer Involvement	Limited, mainly at start & end	Continuous, throughout development
Delivery	Final product delivered at the end	Frequent small releases
Testing	Done after development phase	Done continuously with each iteration
Risk Handling	High risk if requirements change	Low risk due to quick feedback
Best For	Projects with fixed requirements	Projects with evolving requirements

25. Explain the Scrum framework in Agile development.

Scrum Framework is a popular Agile methodology used to manage and deliver complex projects in short, iterative cycles called **sprints** (usually 2–4 weeks). It focuses on collaboration, flexibility, and continuous improvement.

Key Roles in Scrum:

- **Product Owner:** Defines product vision, manages the **Product Backlog**, and prioritizes work.
- **Scrum Master:** Facilitates the Scrum process, removes obstacles, and ensures team follows Agile principles.

- **Development Team:** Cross-functional team that builds and delivers the product increment.

Scrum Artifacts:

- **Product Backlog:** List of all required features and tasks.
- **Sprint Backlog:** Selected tasks to be completed in a sprint.
- **Increment:** Working software delivered at the end of each sprint.

Scrum Events (Ceremonies):

1. **Sprint Planning** – Plan what to deliver in the sprint.
2. **Daily Scrum (Stand-up)** – 15-minute meeting to track progress.
3. **Sprint Review** – Demonstrate completed work to stakeholders.
4. **Sprint Retrospective** – Reflect on process and improve.

26. What are Agile tools? Give examples.

Ans. Agile tools are software or platforms that help teams plan, track, collaborate, and manage Agile projects efficiently. They support **backlogs, sprints, user stories, tasks, and reporting**.

Examples of Agile Tools:

- **Jira** – Widely used for Scrum & Kanban boards, backlog management.
- **Trello** – Simple visual task management using boards & cards.
- **Asana** – Task tracking and team collaboration.
- **Azure DevOps** – For Agile project planning and CI/CD integration.
- **Monday.com** – Visual project management tool.
- **VersionOne** – Enterprise Agile project management.

27. How does Agile handle changing requirements during development?

gile handles changing requirements by using an **iterative and incremental approach**, where work is done in short sprints.

- Changes can be added to the **product backlog** anytime.
- Each sprint allows **reprioritization** based on feedback.
- Continuous customer involvement ensures the product adapts to evolving needs.

28. Compare Agile vs Spiral model.

Ans. Here's a clear **comparison between Agile and Spiral Model**:

Aspect	Agile Model	Spiral Model

Aspect	Agile Model	Spiral Model
Approach	Iterative & incremental with short sprints	Iterative with focus on **risk analysis**
Risk Management	Basic risk handling through feedback	**Strong risk analysis** at every iteration
Flexibility	Highly flexible to changing requirements	Flexible but more structured than Agile
Customer Involvement	Continuous involvement in each sprint	Customer involved at the end of each cycle
Planning	Minimal upfront planning, adaptive planning	Detailed planning for each spiral phase
Delivery	Frequent small releases of working software	Deliverables after each spiral iteration
Best For	Dynamic projects with evolving requirements	Large, high-risk, complex projects

29. What is Pair Programming in Agile?

Ans. Pair Programming in Agile is a practice where **two developers work together on the same code** at one workstation.

- **Driver:** Writes the code.
- **Observer/Navigator:** Reviews the code, thinks about strategy, and suggests improvements.

Benefits: Improves code quality, reduces bugs, enhances knowledge sharing, and increases collaboration.

30. Explain the 4 P's principal People-Product-Process-Project relationship or Discuss the 4 P's of software project management in detail with examples

Ans. The **4 P's of Software Project Management**—**People, Product, Process, and Project**—form the **Management Spectrum**, which ensures successful software development. Let's explain each in detail with examples:

1. People

People are the **most important asset** in any software project. It includes developers, testers, managers, and stakeholders.

- **Role:** Building, managing, and motivating the team.
- **Example:** A skilled Scrum team with developers, a Scrum Master, and a Product Owner working collaboratively.

- **Key focus:** Clear communication, defined roles, and team coordination.

2. Product

Defines **what needs to be built**—its objectives, scope, and requirements.

- **Role:** Understanding customer needs and documenting requirements clearly.
- **Example:** For an online shopping app, the product requirements include catalog management, payment gateway, and user authentication.
- **Key focus:** Correct requirement gathering to avoid rework.

3. Process

The **framework or methodology** used to build the software.

- **Role:** Choosing the right development model (Agile, Waterfall, Spiral, etc.).
- **Example:** Using Agile Scrum for a project with frequently changing requirements.
- **Key focus:** Ensuring quality, consistency, and efficiency in development.

4. Project

The **actual planning and management** of resources, time, and cost to deliver the product.

- **Role:** Scheduling tasks, monitoring progress, and risk management.
- **Example:** Using tools like Jira or MS Project to track sprints and deadlines.
- **Key focus:** Delivering the software **on time, within budget, and meeting quality standards**.

Relationship:

- **People** develop the **Product** by following a proper **Process** within a managed **Project** environment.
- All four are **interconnected**—if one fails, the project suffers.

31. Explain the Product perspective in project management.

Ans. **roduct Perspective** in project management refers to **how the product fits into the overall business environment and interacts with other systems or products.**

It answers **what the product is, why it is needed, and how it will integrate with existing systems.**

Key Points of Product Perspective:

1. **Purpose & Objectives** – Why the product is being developed.

2. **System Interfaces** – How it connects or communicates with other systems.
3. **User Needs** – What users expect from the product.
4. **Constraints** – Hardware, software, regulatory, or business limitations.
5. **Dependencies** – Any external systems or components it relies on.

Example:
If you're building an **online payment gateway**, the product perspective includes:

- Its role in the e-commerce platform
- Interfaces with bank APIs
- Security and regulatory constraints
- Compatibility with mobile and web apps

32. Define Process in the management spectrum.

Ans. In the **Management Spectrum**, **Process** refers to the **framework of activities, methods, and practices used to develop and maintain software efficiently and with quality.**

- It provides **structure and guidelines** for planning, developing, testing, and delivering the product.
- Ensures **consistency, quality, and predictability** in software development.

Example:
Using **Agile Scrum** or **Waterfall model** as the software development process to ensure proper planning, coding, testing, and delivery.

33. What is a Software Project? Give examples.

Ans. A **Software Project** is a planned effort to **develop, maintain, or enhance a specific software product** within defined **time, cost, and quality constraints.**

It involves **requirement analysis, design, coding, testing, deployment, and maintenance** activities to meet user needs.

Examples of Software Projects:

- Developing an **e-commerce website** like Amazon.
- Building a **mobile banking app**.
- Creating an **online learning platform** like Coursera.
- Developing **hospital management software**.

34. What is the purpose of project planning?

The **purpose of project planning** is to **define the project's objectives, scope, schedule, resources, risks, and deliverables** to ensure successful completion.

Key purposes:

1. **Clarify goals & scope** – What needs to be delivered.
2. **Define tasks & responsibilities** – Who will do what.
3. **Estimate time & cost** – Plan budget and schedule.
4. **Identify risks** – Prepare strategies to handle uncertainties.
5. **Ensure smooth execution** – Provide a roadmap for the team.

35. Define Scope and Feasibility Study in software project planning.

Ans. **Scope** in software project planning defines the **boundaries of the project**—what features, functions, and deliverables will (and will not) be included.
Example: For a shopping app, the scope includes user login, product catalog, and payment gateway but excludes warehouse management.

Feasibility Study checks whether the project is **practical and viable** in terms of **technical, economic, legal, and operational aspects** before starting development.
Example: Evaluating if building a new hospital management system is cost-effective and technically possible.

- **Scope** = *What the project will deliver.*
- **Feasibility Study** = *Can the project be successfully done?*

36. Explain Effort Estimation techniques with an example.

Ans. Effort Estimation techniques in software project management are used to **predict the time, cost, and resources needed** to complete a project.

Common Effort Estimation Techniques:

1. **Expert Judgment**
   - Based on the experience of senior developers or managers.
   - *Example:* A senior engineer estimates a login module will take 5 days.
2. **Analogous Estimation**
   - Compare with a similar past project.
   - *Example:* If a previous e-commerce site took 200 hours, a similar one may take about the same.
3. **Top-Down Estimation**
   - Estimate the entire project first, then break it into smaller tasks.

   - *Example:* Total project 500 hours → Login module ~50 hours.
4. **Bottom-Up Estimation**
   - Estimate each small task, then sum up for the total effort.
   - *Example:* Login (20h) + Payment (40h) + Cart (30h) = 90 hours.
5. **Parametric Estimation**
   - Use mathematical models (e.g., cost per line of code or function points).
   - *Example:* If 1 function point = 5 hours, and project has 100 FP → 500 hours.

37. What is Function Point Analysis (FPA)?

Ans. Function Point Analysis (FPA**)** is a **software sizing technique** used to measure the **functionality** delivered to the user, independent of the programming language or technology.

It helps in **estimating effort, cost, and time** for software development.

Key Idea:

- It counts **user-visible functions** like inputs, outputs, user interactions, files, and interfaces.
- Each function is given a weight (simple, average, or complex).
- Total function points are calculated and used for effort estimation.

Example:
An online banking app may have:

- **External Inputs (EI):** Login, Fund Transfer
- **External Outputs (EO):** Transaction Report
- **External Inquiries (EQ):** Balance Inquiry
- **Internal Logical Files (ILF):** Customer Database
- **External Interface Files (EIF):** Bank APIs

38. What is the COCOMO Model for effort estimation?

Ans. The **COCOMO Model (Constructive Cost Model)** is a **software effort estimation model** developed by Barry Boehm. It predicts the **effort, time, and cost** to develop a software project based on the project size (measured in **KLOC – Thousands of Lines of Code**).

COCOMO Formula:

Effort (person-months)=a×(KLOC)b\text{Effort (person-months)} = a \times (\text{KLOC})^bEffort (person-months)=a×(KLOC)b
Development Time (months)=c×(Effort)d\text{Development Time (months)} = c \times (\text{Effort})^dDevelopment Time (months)=c×(Effort)d

Where **a, b, c, d** are constants depending on the project type:

- **Organic:** Simple, small team, familiar environment.
- **Semi-Detached:** Medium complexity, mixed experience.
- **Embedded:** Complex, tight constraints.

Example:
If a project has **32 KLOC** and is **organic**,

Effort=2.4×(32)1.05≈90 person-months\text{Effort} = 2.4 \times (32)^{1.05} \approx 90 \text{ person-months}Effort=2.4×(32)1.05≈90 person-months

COCOMO estimates **effort and time** using **project size (KLOC)** and project type (organic, semi-detached, embedded).

39. Define schedule and staffing in a project.

Ans. Schedule in a project defines the **timeline of activities**, showing **when tasks start, when they end, and their sequence** to ensure the project is completed on time.
Example: A Gantt chart showing design (Week 1–2), coding (Week 3–6), and testing (Week 7–8).

Staffing refers to **assigning the right number of people with the required skills** to different project tasks at the right time.
Example: 2 developers for frontend, 1 tester for QA, 1 project manager for monitoring.

- **Schedule** = *When tasks happen.*
- **Staffing** = *Who will do the tasks and how many are needed.*

40. Why is team management important in software projects?

Ans. Team management is important in software projects because it ensures the team works **efficiently, collaboratively, and towards common goals** to deliver quality software on time.

Key Reasons:

1. **Improves communication** – Avoids misunderstandings and delays.
2. **Enhances productivity** – Proper role allocation maximizes output.
3. **Builds motivation & trust** – Keeps the team engaged and motivated.
4. **Resolves conflicts** – Handles issues quickly to maintain harmony.
5. **Ensures timely delivery** – Coordinated efforts lead to on-time completion.
6. **Maintains quality** – Well-managed teams follow processes and standards.

41. What is Risk Management in software projects?

Ans. Risk Management in software projects is a systematic process of identifying, analyzing, planning, and responding to potential risks that may affect the successful completion of the project. In software development, risks can arise from various sources such as unclear requirements, technology changes, resource limitations, budget constraints, unrealistic schedules, or external factors like market shifts and legal issues.

The main goal of risk management is to minimize the probability and impact of negative events (threats) while maximizing opportunities. It typically involves the following steps:

1. **Risk Identification** – Recognizing potential risks that may affect the project, like technical failures, lack of skilled resources, or requirement changes.
2. **Risk Analysis** – Assessing the likelihood and potential impact of each identified risk.
3. **Risk Prioritization** – Ranking risks based on their severity and probability to focus on the most critical ones.
4. **Risk Mitigation Planning** – Creating strategies to avoid, reduce, transfer, or accept risks.
5. **Risk Monitoring and Control** – Continuously tracking risks, reviewing mitigation actions, and updating the risk management plan throughout the project lifecycle.

Effective risk management improves decision-making, reduces project delays, controls costs, enhances software quality, and increases the chances of delivering a successful project.

42. Explain Risk Identification and Assessment with examples.

Ans. Risk Identification

Risk Identification is the process of recognizing and documenting potential risks that may affect the project's cost, schedule, quality, or performance. It involves brainstorming, using past project checklists, expert judgment, and analyzing requirements and project plans.

Examples:

- Changing or unclear client requirements (**scope risk**)
- Using new or untested technology (**technical risk**)
- Shortage of skilled team members (**resource risk**)
- Delay in hardware/software delivery (**schedule risk**)
- Budget overruns due to additional features (**cost risk**)

Risk Assessment

Risk Assessment involves analyzing and prioritizing the identified risks based on:

- **Probability** (how likely it is to occur)

- **Impact** (how severe the effect would be if it occurs)

A **Risk Matrix** is often used to classify risks as low, medium, or high.

Example:

- Requirement changes → **High probability, High impact** → Critical risk
- New technology failure → **Medium probability, High impact** → High risk
- Delay in feedback → **Low probability, Medium impact** → Medium ris

43. What is Risk Control?

Risk Control is the process of implementing actions to minimize, eliminate, or manage the impact of identified risks on a software project. It involves monitoring risks, executing mitigation plans, and taking corrective measures to keep the project on track.

44. Define Project Monitoring Plan.

Ans. A **Project Monitoring Plan** is a detailed framework that outlines how a project's activities, progress, and performance will be continuously tracked and evaluated throughout its lifecycle. It defines **what will be monitored**, **how it will be measured**, **who is responsible**, and **how often monitoring will occur**.

It typically includes:

- Key performance indicators (KPIs)
- Schedule and budget tracking methods
- Quality and risk monitoring approaches
- Reporting frequency and communication methods

Example: In a software project, the monitoring plan may include tracking completed modules, defect rates, resource utilization, and milestone achievements to ensure the project stays aligned with its goals.

45. What is Detailed Scheduling and why is it important?

Ans. Detailed Scheduling in software project management refers to the process of **breaking down the overall project plan into smaller, well-defined tasks with specific timelines, resource allocations, dependencies, and milestones**. It provides a **clear roadmap** of *who will do what, when, and how*.

Why is Detailed Scheduling Important?

1. **Better Time Management** – Ensures tasks are completed on time, reducing delays.
2. **Resource Optimization** – Helps allocate people, tools, and budget efficiently.

3. **Improved Coordination** – Clarifies dependencies and avoids conflicts between tasks.
4. **Progress Tracking** – Makes it easier to monitor project progress and adjust when needed.
5. **Risk Reduction** – Identifies potential bottlenecks early.
6. **Stakeholder Confidence** – Provides transparency and realistic timelines.

Example:
In a software development project, detailed scheduling might specify:

- *UI design – 5 days (John)*
- *Backend API development – 10 days (Sara, depends on UI completion)*
- *Testing – 7 days (QA team, starts after API)*

46. What is the W5HH principle? Explain7 questions of W5HH principle

Ans. The **W5HH Principle** is a project management approach introduced by **Barry Boehm**. It is used to **plan, organize, and track software projects** by answering **7 key questions** related to project objectives, constraints, and responsibilities.

It's called **W5HH** because it has **5 W-questions** and **2 H-questions**.

1. **Why is the system being developed?**
   - Defines the **project objectives, business needs, and expected benefits**.
   - Example: *Why are we building an online shopping app? → To improve customer reach and sales.*
2. **What will be done?**
   - Specifies the **scope**, **deliverables**, and **key features**.
   - Example: *What will be done? → Website with product catalog, payment gateway, and user accounts.*
3. **When will it be done?**
   - Deals with **project timelines, milestones, and scheduling**.
   - Example: *When will it be done? → UI in 2 weeks, backend in 4 weeks, testing in 2 weeks.*
4. **Who is responsible for a function?**
   - Defines **roles, responsibilities, and ownership**.
   - Example: *Who is responsible? → John for UI, Sara for backend, QA team for testing.*
5. **Where are they organizationally located?**
   - Shows the **team structure** and **communication hierarchy**.
   - Example: *Developers in the IT department, QA in the testing team.*
6. **How will the job be done technically and managerially?**
   - Explains **methods, tools, technologies, and processes** to be used.
   - Example: *How technically? → Using Agile, GitHub, and ReactJS.*
7. **How much of each resource is needed?**
   - Defines the **effort, cost, and resource estimation**.
   - Example: *How much? → 5 developers, 2 testers, $20,000 budget.*

Why is W5HH Important?

- Ensures **clarity before starting the project**
- Avoids confusion about **scope, time, cost, and responsibilities**
- Improves **communication and planning**

47. What are the key design concepts in software engineering? Explain abstraction and modularity with example.

Ans. In software engineering, **design concepts** help create systems that are **easy to understand, maintain, and extend**. The main concepts include **abstraction, modularity, architecture, information hiding, refinement, separation of concerns, and design patterns**.

Let's explain **abstraction** and **modularity** in medium detail.

---

1. Abstraction

- **Meaning:**
  Abstraction focuses on **highlighting essential features** of a system while **hiding unnecessary details**.
- **Purpose:**
  It reduces complexity and allows the designer to work at a **higher level of understanding** without worrying about low-level details.
- **Example:**
  In a **banking application**, a `withdraw(amount)` function hides complex details like database queries, balance checks, and transaction logs. The user or programmer only sees the main functionality – withdrawing money.

2. Modularity

- **Meaning:**
  Modularity is the practice of **dividing a software system into smaller, independent, and manageable modules**.
- **Purpose:**
  It makes the software **easier to develop, test, and maintain** because changes in one module have minimal impact on others.
- **Example:**
  In an **e-commerce application**, you can have separate modules for *User*

Authentication, *Product Catalog*, *Shopping Cart*, and *Payment Gateway*. Each module can be developed and tested independently but works together to form the complete system.

48. What is Requirements Specification?

Ans. Requirements Specification is a **formal document** that clearly defines **what the software system should do**. It describes **functional requirements (what the system must perform)** and **non-functional requirements (quality attributes like performance, security, usability)** in a **structured and detailed way**.

Key Points

- Acts as an **agreement** between stakeholders and developers.
- Removes ambiguity by **clearly stating expected system behavior**.
- Serves as a **reference for design, development, and testing**.

Types of Requirements in Specification

1. **Functional Requirements** – Specific features and functions
    - Example: *The system shall allow users to log in using email and password.*
2. **Non-Functional Requirements** – Quality attributes
    - Example: *The system shall respond within 2 seconds.*

Example

For an **online banking system**:

- *Functional:* Users can transfer funds between accounts.
- *Non-functional:* The system should be available 99.9% of the time.

So, **Requirements Specification = Clear, structured, and detailed description of what the software will do and how it should behave.**

49. What is the importance of Use Cases in requirements?

Ans. A **Use Case** describes **how a user (actor) interacts with the system** to achieve a specific goal. In requirements engineering, **use cases are very important** because they help in **understanding, clarifying, and validating requirements**.

Why Use Cases are Important?

1. **Clarify Functional Requirements**
   - Show *what the system should do* from the user's perspective.
   - Example: *"Login Use Case" describes how a user enters credentials and gets access.*
2. **Improve Communication**
   - Help stakeholders, developers, and testers have a **common understanding** of system behavior.
3. **Identify Missing Requirements**
   - By modeling real interactions, you discover **additional scenarios and edge cases**.
4. **Basis for Design & Testing**
   - Designers can derive **UI flows** and **architecture** from use cases.
   - Testers can create **test cases** directly from them.
5. **Visual Representation**
   - Use Case Diagrams provide a **simple, easy-to-understand picture** of how users interact with the system.

Example

For an **online shopping app**:

- **Actors:** Customer, Admin
- **Use Cases:** Browse Products, Add to Cart, Checkout, Make Payment, Manage Products

50. What is a Design Model? List and explain its major elements.

Ans. A **Design Model** in software engineering is a **blueprint of the system** that describes *how the software will be structured, organized, and implemented* to satisfy the requirements.

It transforms **requirements (what the system should do)** into a **design (how the system will do it)**.

Major Elements of a Design Model

1. **Data Design**
   - Defines **how data is stored, organized, and accessed**.
   - Example: *Database schemas, data structures, and data flow between components.*
2. **Architectural Design**
   - Represents the **overall structure** of the system.
   - Shows **major components (modules), their interactions, and communication patterns**.
   - Example: *Client-server architecture, layered architecture.*
3. **Interface Design**
   - Specifies **how users and other systems will interact with the software**.

- Includes **user interface (UI), APIs, and external system interfaces**.
- Example: *Login screen design, REST API endpoints.*

4. **Component-Level Design**
   - Focuses on the **internal logic of individual modules or components**.
   - Includes **class diagrams, algorithms, and control structures**.
   - Example: *Designing the logic for the payment processing module.*
5. **Deployment Design (Optional)**
   - Shows **how the software will be deployed in the physical environment**, including **servers, networks, and hardware nodes**.

51. Differentiate between Functional and Non-functional requirements.

Ans. Difference between Functional and Non-Functional Requirements

Aspect	Functional Requirements	Non-Functional Requirements
Definition	Define **what the system should do** – specific behaviors, features, or functions.	Define **how the system should behave** – quality attributes and constraints.
Focus	Focus on **system functionality**.	Focus on **system performance, usability, and quality**.
Nature	Describes **actions or services** provided by the system.	Describes **properties, constraints, or quality attributes**.
Examples	□ User login □ Fund transfer in banking □ Generating reports	□ System must respond within 2 seconds □ Availability should be 99.9% □ Must support 1,000 users simultaneously
Verification	Verified by **functional testing** (e.g., black-box testing, use case testing).	Verified by **performance, security, usability, and reliability testing**.
Change Impact	Changes may affect **specific modules**.	Changes may affect **entire system architecture or infrastructure**.

52. What is Requirements Validation?

Ans. Requirements Validation is the process of **checking and ensuring that the documented requirements are correct, complete, consistent, and meet the actual needs of stakeholders.**

why is Requirements Validation Important?

- Detects **errors and ambiguities early** (before design & development).
- Saves **time, cost, and effort** by avoiding rework later.
- Ensures **stakeholder expectations are correctly captured**.

Common Techniques for Requirements Validation

1. **Reviews/Inspections** – Stakeholders review the SRS document.
2. **Prototyping** – Build a mock-up to confirm understanding.
3. **Model Validation** – Use use-case diagrams or models to verify flows.
4. **Test Case Generation** – Derive test cases from requirements to see if they are testable.
5. **Stakeholder Meetings/Workshops** – Directly validate requirements with users.

53. Explain Requirements Analysis in detail.

Requirements Analysis is the process of **gathering, understanding, refining, and documenting the needs of stakeholders** for a software system.

It converts **high-level business needs** into **detailed, clear, and structured requirements** that can be used for design and development.

Objectives of Requirements Analysis

- Identify **functional and non-functional requirements**
- Ensure requirements are **complete, consistent, and unambiguous**
- Understand **system boundaries and constraints**
- Resolve **conflicts among stakeholders**

Key Activities in Requirements Analysis

1. **Requirements Elicitation**
   - Gathering requirements from stakeholders using interviews, surveys, observations, brainstorming, etc.
2. **Requirements Modeling**
   - Representing requirements using **use case diagrams, data flow diagrams (DFD), or UML models** for better understanding.
3. **Requirements Categorization**

   - Dividing into **functional, non-functional, and domain requirements**.
4. **Conflict Resolution**
   - Resolving inconsistencies or contradictions among stakeholder needs.
5. **Feasibility Analysis**
   - Checking if the requirements are **technically and economically feasible**.
6. **Requirements Prioritization**
   - Ranking requirements based on **importance, cost, and impact**.
7. **Requirements Documentation**
   - Preparing an **SRS (Software Requirements Specification)** document.

Example

For an **Online Food Delivery App**:

- **Elicitation:** Talk to customers, restaurant owners, and delivery agents.
- **Modeling:** Create use cases like *Browse Menu*, *Place Order*, *Track Delivery*.
- **Categorization:**
  - *Functional:* Place order, make payment
  - *Non-functional:* Must respond within 2 seconds
- **Feasibility:** Check if real-time GPS tracking is possible within budget.
- **Prioritization:** Login & ordering features are more critical than reviews section.

54. Why is Requirements Gathering important?

Ans. Requirements Gathering is important because it:

- Helps **understand stakeholder needs** clearly.
- Defines the **project scope** to avoid confusion.
- Reduces **ambiguity and misunderstandings**.
- Saves **time and cost** by avoiding rework.
- Improves **planning, estimation, and quality**.
- Provides a **baseline for validation and testing**.

55. What are the challenges in Requirements Elicitation?

Ans. Challenges in Requirements Elicitation:

- **Unclear or vague requirements** from stakeholders
- **Changing business needs** during the project
- **Communication gaps** and technical language issues
- **Conflicting stakeholder demands**
- **Limited time and stakeholder availability**

56. Define Stakeholder Analysis in requirement engineering.

Ans. Stakeholder Analysis is a key activity in requirement engineering where all individuals or groups who have an interest in the software system are **identified, categorized, and analyzed**. It focuses on understanding their **roles, needs, expectations, influence, and priority** in the project.

This process helps in:

- Identifying **who will use, manage, or be affected** by the system.
- Understanding **conflicting needs** between different stakeholders.
- Prioritizing requirements based on **stakeholder importance and influence**.

57. What is Feasibility Analysis in requirements engineering?

Ans. **Feasibility Analysis** in requirements engineering is the process of **evaluating whether the proposed software system is practical, achievable, and worth developing** considering different constraints. It checks if the project can be successfully completed within the **time, budget, technology, and resource limits** while meeting stakeholder needs.

Types of Feasibility

1. **Technical Feasibility** – Can the required technology and expertise support the system?
2. **Economic Feasibility** – Is the project cost-effective and within budget?
3. **Operational Feasibility** – Will the system work smoothly in the organization and be accepted by users?
4. **Schedule Feasibility** – Can it be completed within the required timeframe?
5. **Legal Feasibility** – Does it comply with laws, regulations, and policies?

58. Explain Requirements Management and its benefits.

Ans. Requirements Management is the process of **documenting, organizing, tracking, and controlling changes to requirements throughout the software development lifecycle** to ensure they remain clear, consistent, and aligned with stakeholder needs.

Benefits

- Ensures **requirements are traceable and up to date**
- Helps manage **changes effectively**
- Improves **communication between stakeholders and developers**
- Reduces **project risks and rework**
- Ensures the final system meets **business goals**

59. What is the role of prototyping in requirements validation?

Ans. **Prototyping** helps in requirements validation by creating a **working model or mock-up** of the system, allowing stakeholders to **see and interact with it early**.

This helps to:

- **Clarify and refine requirements**
- Detect **missing or incorrect requirements**
- Improve **stakeholder understanding and feedback**
- Reduce **ambiguity and misunderstandings**

60. Differentiate between User Requirements and System Requirements.

Ans. Difference between User Requirements and System Requirements

Aspect	User Requirements	System Requirements
Definition	High-level needs and expectations of end users, described in simple language.	Detailed description of system functions, features, and constraints.
Audience	Intended for **users and stakeholders** (non-technical).	Intended for **developers, designers, and testers** (technical).
Detail Level	General and abstract.	Precise and detailed.
Format	Natural language, use cases, or diagrams.	Structured document like **SRS** with functional & non-functional details.
Example	*"The user should be able to view account balance easily."*	*"The system shall display the account balance by fetching data from the database within 2 seconds."*

61. Explain Process Layer in software engineering layered technology.

Ans. In **software engineering layered technology**, the **Process Layer** sits **above the Quality Focus layer** and provides the **framework for applying software engineering methods and tools effectively**.

It defines the **software development process** that guides how software is planned, developed, and maintained.

Key Points about Process Layer

- It establishes the **activities, tasks, and workflows** for software development.
- Includes **communication, planning, modeling, construction, and deployment** activities.
- Ensures **consistency and coordination** across the project.
- Supports **process models** like **Waterfall, Agile, Spiral, Incremental, etc.**
- Acts as a **bridge** between **quality focus** and **methods/tools**.

62. What is the Method Layer? Give an example.

Ans. The **Method Layer** provides the **technical methods** used to build software, such as **requirements analysis, design methods, coding techniques, testing strategies, and maintenance approaches**.

Example:

- **Object-Oriented Design (OOD)** for designing classes and objects.
- **Structured Analysis and Design Method (SADM)** for modeling system structure and behavior.

63. What is the importance of CASE tools?

Ans. CASE (Computer-Aided Software Engineering) tools are software applications that **support and automate various activities of software development**, such as analysis, design, coding, testing, and maintenance.

Example:

- **Rational Rose** for UML modeling
- **JIRA** for project tracking
- **Selenium** for automated testing

CASE tools improve ***efficiency, quality, and collaboration*** in software engineering by automating and managing development tasks.

64. What are the key design concepts in software engineering? Explain abstraction and modularity with example.

Ans. Key design concepts include **abstraction, modularity, architecture, information hiding, refinement, separation of concerns, and design patterns.**

- **Abstraction** → *Hides unnecessary details and shows only essential features.*
  Example: A `print()` function hides low-level details of how text is displayed.
- **Modularity** → *Divides the system into smaller, independent modules for easier development and maintenance.*
  Example: In an e-commerce app, separate modules for *Login, Product Catalog, and Payment*.

65. What is the relationship between process, methods, and tools?

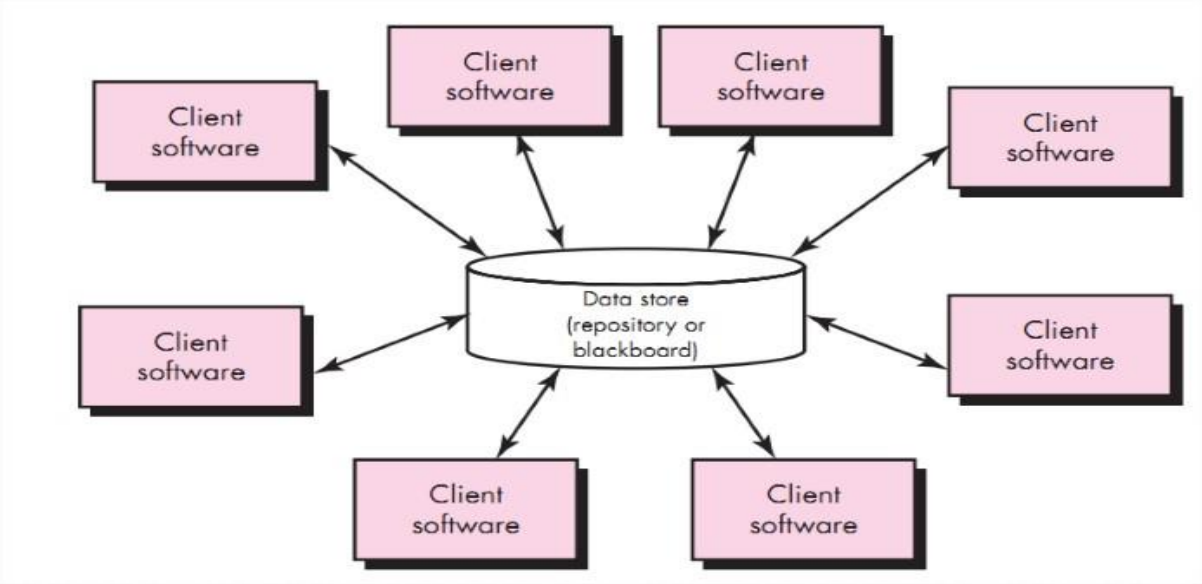
Ans. Relationship between Process, Methods, and Tools:

- **Process** defines the **framework of activities** (like planning, design, coding, testing) to build software.
- **Methods** provide the **technical approach** to perform those activities (like object-oriented design, testing techniques).
- **Tools** support and **automate methods and processes** (like UML tools, testing tools).

*Process is the **framework**, methods are the **techniques**, and tools are the **supporting aids** to implement them.*

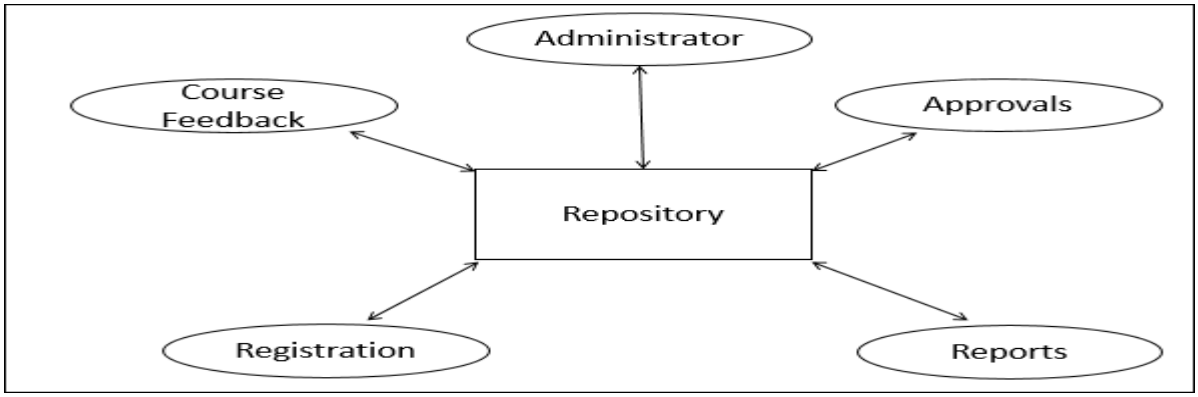
66. Explain data centered architecture.

Ans. Data-Centered Architecture is a software architectural style where a **central data store (repository)** is the main component, and **different software components (clients)** interact with it to retrieve or update data.



Key Features

- **Central Repository:** A single data source stores all system data.
- **Independent Components:** Components communicate **only through the central data store**, not directly with each other.
- **Loose Coupling:** Changes in one component don't affect others as long as the data format remains consistent.



Types

1. **Database-Centered Architecture** – Uses a database as the repository.
2. **Blackboard Architecture** – Components post data to a common blackboard and react to changes.

Example

- **Banking System:** A central database stores all account information; services like *fund transfer, loan processing, and reporting* access the same repository.
- **GitHub:** Central repository stores code, and different users interact with it.

67. What are the different categories of software tools?

Ans. Different Categories of Software Tools

1. **Project Management Tools** – Help in planning and tracking (e.g., JIRA, Trello).
2. **Modeling and Design Tools** – Create diagrams and models (e.g., UML tools, Rational Rose).
3. **Code Generation Tools** – Assist in writing and generating code (e.g., Eclipse, Visual Studio).
4. **Testing Tools** – Automate and manage testing (e.g., Selenium, JUnit).
5. **Configuration Management Tools** – Manage versions and changes (e.g., Git, SVN).
6. **Documentation Tools** – Help create and maintain project documents (e.g., Doxygen, MS Word).

68. What is Software Quality Assurance (SQA) tools' role?

Ans. SQA tools help ensure the **quality, reliability, and correctness** of software by supporting various quality-related activities throughout the development process.

Key Roles

- **Automate Testing** → Run unit, integration, and regression tests (e.g., Selenium, JUnit).
- **Track Defects** → Manage and monitor bugs/issues (e.g., Bugzilla, JIRA).
- **Measure Code Quality** → Check coding standards, complexity, and security (e.g., SonarQube).
- **Ensure Process Compliance** → Verify adherence to standards like ISO, CMMI.
- **Support Continuous Integration** → Integrate quality checks in CI/CD pipelines (e.g., Jenkins).

69. Explain why software projects often fail despite good planning.

Ans. Why Software Projects Often Fail Despite Good Planning

- **Changing or unclear requirements** during development
- **Poor communication** between stakeholders and team
- **Inadequate resources or skills**
- **Unrealistic time and cost estimates**
- **Scope creep** (uncontrolled addition of new features)
- **Lack of proper risk management**
- **Weak testing and quality control**

70. What are the key differences between Predictive and Adaptive models?

Ans. Differences between Predictive and Adaptive Models

Aspect	Predictive Model	Adaptive Model
Approach	Plan-driven, follows a **fixed sequence** of phases.	Change-driven, **adjusts to evolving requirements**.
Requirements	Assumes requirements are **stable and well-defined** early.	Suitable for **uncertain or changing requirements**.
Flexibility	Less flexible, harder to accommodate changes.	Highly flexible and iterative.
Examples	Waterfall, V-Model	Agile, Spiral, Scrum

71. Discuss the role of communication in software project success.

Ans. Role of Communication in Software Project Success

- Ensures **clear understanding of requirements** among stakeholders and developers.
- Helps in **resolving conflicts** and aligning team goals.
- Facilitates **timely updates** on progress, risks, and changes.
- Improves **coordination between team members**.
- Builds **trust and transparency** with clients.

72. What are the ethical issues in software engineering?

Ans. Ethical Issues in Software Engineering

- **Privacy Violations** – Misusing or leaking user data.
- **Security Risks** – Developing insecure software intentionally or negligently.
- **Intellectual Property Theft** – Copying code or software illegally.
- **Software Reliability** – Delivering faulty software that can cause harm.
- **Misleading Clients** – Hiding limitations or overpromising features.
- **Bias and Fairness** – Creating software that discriminates or is biased.

73. Explain how risk management improves project success.

isk management is the process of **identifying, analyzing, and mitigating potential risks** that could negatively impact a software project. It improves project success by:

1. **Early Identification of Problems**
   - Detects risks (e.g., budget overruns, delays, technical failures) before they become critical.
2. **Better Planning & Preparedness**
   - Creates **contingency plans** to handle unexpected issues smoothly.
3. **Reduces Uncertainty**
   - Provides a clearer view of potential challenges, making decision-making easier.
4. **Minimizes Project Failures**
   - Proactively resolving risks lowers the chance of missing deadlines or exceeding costs.
5. **Improves Stakeholder Confidence**
   - Shows that the project team is **prepared and reliable**, increasing trust.

74. Give a case study of Waterfall model failure.

Ans. The project followed the **Waterfall Model**, with fixed sequential phases: requirements → design → implementation → testing → deployment.

Reasons for Failure

1. **Changing Requirements**
   - FBI requirements kept evolving after the initial requirement phase, but Waterfall was rigid and couldn't adapt.
2. **Late Testing & Feedback**
   - Testing was done **only after full development**, revealing major issues too late.
3. **Poor Communication**
   - End-users (FBI agents) were not actively involved after the requirement phase, leading to a system that didn't meet their needs.
4. **High Cost & Delay**
   - The project cost **$170 million** but was **abandoned after 3 years** without delivering a usable system.

75. Discuss advantages of evolutionary models over traditional models.

Ans. Advantages of Evolutionary Models over Traditional Models

- **Handles changing requirements** easily with iterative development.
- Provides **early delivery of working software** for user feedback.

- **Reduces risk** by developing in small, manageable increments.
- Improves **customer satisfaction** through continuous involvement.
- Easier to **detect and fix issues early** compared to late testing in traditional models.

76. What is the impact of team management on software quality?

Ans. Impact of Team Management on Software Quality

- Good team management ensures **clear roles and responsibilities**, reducing errors.
- Enhances **communication and collaboration**, leading to better understanding of requirements.
- Motivates the team, improving **productivity and attention to quality**.
- Helps in **conflict resolution** and smooth workflow.
- Ensures proper **planning, monitoring, and adherence to quality standards**.

77. What are measurable project metrics in software engineering?

Ans. Measurable Project Metrics in Software Engineering are **quantitative indicators** used to evaluate the progress, quality, productivity, and performance of a software project. They help in **monitoring, controlling, and improving** the software development process.

Key Measurable Project Metrics

1. **Size Metrics**
   - Measure the size of the software product.
   - Examples:
     - **Lines of Code (LOC)**
     - **Function Points (FP)**
     - Number of modules, classes, or interfaces
2. **Effort Metrics**
   - Measure the amount of work required.
   - Examples:
     - Person-hours or person-days
     - Effort spent on coding, testing, etc.
3. **Schedule Metrics**
   - Measure project timeline adherence.
   - Examples:
     - Planned vs. actual completion time
     - Milestone achievement percentage
4. **Cost Metrics**
   - Measure project cost performance.
   - Examples:
     - Planned vs. actual budget
     - Cost per function point or LOC
5. **Quality Metrics**

- Measure the quality of the software.
- Examples:
  - Defect density (defects/LOC or FP)
  - Mean time to failure (MTTF)
  - Customer-reported issues

6. **Productivity Metrics**
   - Measure how efficiently resources are used.
   - Examples:
     - LOC or FP per person-month
     - Defects fixed per day
7. **Risk Metrics**
   - Measure project risks and their impact.
   - Examples:
     - Number of high-risk items open
     - Risk exposure levels

Why Are These Metrics Important?

- Help in **tracking project progress**
- Support **better estimation and planning**
- Ensure **quality control**
- Aid in **decision-making and process improvement**

78. Give an example of a Risk Matrix in software development.

A **Risk Matrix** in software development shows the **likelihood** of a risk happening versus its **impact** on the project.

Example Risk Matrix

Impact ↓ / Likelihood →	**Low**	**Medium**	**High**
High Impact	Medium	High	Critical
Medium Impact	Low	Medium	High
Low Impact	Low	Low	Medium

Example:

- *Risk:* Delay in third-party API integration
  - **Likelihood:** High
  - **Impact:** High
  - **Matrix Level: Critical**

79. What is the role of a project manager in software projects?

Ans. The **role of a Project Manager** in software projects is to **plan, organize, lead, and control** the project to ensure it meets its goals within **scope, time, cost, and quality constraints**.

Key Responsibilities (Short):

- **Planning:** Define project scope, schedule, and resources.
- **Team Management:** Assign tasks, motivate, and resolve conflicts.
- **Risk Management:** Identify and mitigate risks.
- **Communication:** Coordinate between stakeholders and the team.
- **Monitoring & Control:** Track progress, budget, and quality.

80. How does time-boxing help in Agile?

Ans. In **Agile**, **time-boxing** means fixing a **maximum time limit** for an activity (like a sprint, meeting, or task).

How it helps:

- Keeps work **focused and prioritized**
- Prevents **scope creep**
- Ensures **faster delivery** in small increments
- Improves **predictability and productivity**

Example: A **2-week sprint** is a time-box for completing selected user stories.

81. What is the importance of documentation in software projects.

Ans. Documentation in software projects is important because it provides a **clear reference** for understanding, maintaining, and improving the software.

Importance:

1. **Knowledge Sharing** – Helps new team members understand the system easily.
2. **Maintenance & Support** – Guides future bug fixes, updates, and enhancements.
3. **Communication** – Acts as a common reference for developers, testers, clients, and stakeholders.
4. **Legal & Compliance** – Ensures contracts, licenses, and regulatory requirements are met.
5. **Quality Assurance** – Defines requirements, design, and test cases clearly to avoid misunderstandings.

Example: **Requirement Specification Document (SRS)** ensures the team builds exactly what the client needs.

82. What is an *Entity-Relationship (E-R) diagram*? Explain different types of *attributes* in E-R diagrams and draw an E-R diagram for library management.

An **E-R Diagram** is a **graphical representation** of the data and its relationships in a database. It shows:

- **Entities** (real-world objects like *Student, Book*)
- **Attributes** (properties of entities like *Name, ID*)
- **Relationships** between entities (like *borrows, issues*).

It helps in designing the **logical structure of a database**.

Types of Attributes in E-R Diagrams

1. **Simple Attribute**
   - Cannot be divided further.
   - Example: `Name`, `RollNo`.
2. **Composite Attribute**
   - Can be split into smaller sub-parts.
   - Example: `FullName` → `FirstName + LastName`.
3. **Derived Attribute**
   - Value can be **derived** from other attributes.
   - Example: `Age` can be derived from `DateOfBirth`.
4. **Multi-valued Attribute**
   - Can have **multiple values** for one entity.
   - Example: `PhoneNumbers` for a person.
5. **Key Attribute**
   - Uniquely identifies an entity.
   - Example: `StudentID`, `BookID`.

E-R Diagram for Library Management System

Entities & Relationships:

- **Student** *(StudentID, Name, Dept)*
- **Book** *(BookID, Title, Author)*
- **Librarian** *(LibrarianID, Name)*
- **Issue** *(IssueDate, ReturnDate)*
- Relationships:
  - Student **borrows** Book
  - Librarian **issues** Book

84. Define project scope.

Ans. Project scope is the detailed description of the project's objectives, deliverables, tasks, timelines, and boundaries. It defines what work is to be done, what is not included, and ensures that all stakeholders have a common understanding of the expected outcomes.

83. What is the importance of defining scope in a software project?

Ans. Defining the scope is **crucial** because it:

1. **Clarifies objectives** – Ensures all stakeholders understand what the software will deliver.
2. **Prevents scope creep** – Avoids unnecessary changes or additions beyond the planned work.
3. **Helps in accurate planning** – Supports better estimation of time, cost, and resources.
4. **Improves communication** – Aligns the development team and client expectations.
5. **Sets clear boundaries** – Defines what is included and excluded from the project.
6. **Ensures quality delivery** – Focuses efforts on agreed deliverables, reducing rework.

84. Differentiate between scope and objectives of a project.

Ans. Difference between Project Scope and Project Objectives

Aspect	Project Scope	Project Objectives
Meaning	Defines the **boundaries**, deliverables, and work required in the project.	Defines the **specific goals** the project aims to achieve.
Focus	Focuses on **what work will be done** and what is excluded.	Focuses on **why the project is being done** and the desired outcomes.
Nature	Descriptive – explains the **extent of work**.	Measurable – defines **success criteria** (e.g., cost, time, quality).
Example	*Developing a mobile banking app with login, balance check, and transfer features.*	*Improve user convenience by 50% and reduce transaction time by 30%.*
Timeline	Helps in defining **tasks and deliverables** during execution.	Guides the **overall direction** and success measurement of the project.

85. What are the main components of a feasibility study?

Ans. A feasibility study includes technical, economic, operational, legal, and schedule feasibility to evaluate whether the project is practical, cost-effective, and achievable within time and constraints.

Components of a Feasibility Study

1. **Technical Feasibility** – Checks whether the required technology, tools, and expertise are available to develop the project.
2. **Economic Feasibility** – Evaluates cost vs. benefits to see if the project is financially viable.
3. **Operational Feasibility** – Assesses whether the system will work smoothly in the existing environment and meet user needs.
4. **Legal Feasibility** – Ensures the project complies with legal, regulatory, and licensing requirements.
5. **Schedule Feasibility** – Determines if the project can be completed within the required time frame.

86. Explain the relationship between scope definition and project cost estimation.

Ans. Relationship between Scope Definition and Project Cost Estimation

- **Scope definition** outlines **what work will be done, deliverables, and boundaries** of the project.
- **Project cost estimation** depends directly on the scope, as it determines the **amount of work, required resources, time, and complexity**.

87. What is effort estimation in software project management?

Ans. Effort estimation is the process of predicting the **amount of time and human resources (person-hours or person-days)** required to complete a software project or a specific task.

It helps in:

- **Planning schedules and budgets**
- **Allocating resources effectively**
- **Avoiding delays and cost overruns**

Example:
If developing a module requires 2 developers working for 5 days, the **effort = 2 × 5 = 10 person-days**.

88. Why is effort estimation important?

Ans. Effort estimation is important because it helps in accurate project planning, scheduling, budgeting, resource allocation, and prevents delays or cost overruns.

89. List three popular effort estimation techniques.

Ans. Three Popular Effort Estimation Techniques:

Software Engineering (303105253)

1. Lines of Code (LOC) based estimation
2. Function Point Analysis (FPA)
3. COCOMO (Constructive Cost Model)

1. **Lines of Code (LOC) Based Estimation**
   - Estimates effort by predicting the total **number of lines of code** the software will have.
   - More LOC → More effort required.
   - **Limitation:** Depends on coding style and doesn't consider functionality.
2. **Function Point Analysis (FPA)**
   - Estimates effort based on the **functionality delivered to the user** (inputs, outputs, files, interfaces).
   - Independent of programming language or technology.
   - More **function points** → higher effort.
3. **COCOMO (Constructive Cost Model)**
   - A mathematical model developed by **Barry Boehm** to estimate **effort, cost, and schedule** based on project size (in KLOC – thousand lines of code).
   - Has three levels: **Basic, Intermediate, and Detailed COCOMO**.
   - Example formula: *Effort = a × (KLOC)^b*

- **LOC** → counts code size.
- **FPA** → measures functionality.
- **COCOMO** → mathematical model using project size.

90. What is the COCOMO model?

Ans. Above ans.

91. Differentiate between **COCOMO Basic, Intermediate, and Detailed** models

Ans. Difference between COCOMO Basic, Intermediate, and Detailed Models

Model	Description	Factors Considered	Accuracy
Basic COCOMO	Provides a **quick effort estimate** based only on project size (KLOC).	Only considers **Lines of Code (KLOC)**.	Least accurate
Intermediate COCOMO	Improves estimation by adding **cost drivers** like product complexity, team capability, tools, etc.	KLOC + **15 cost drivers**.	More accurate than Basic
Detailed COCOMO	Extends Intermediate model by analyzing effort for **each software component (module)** separately.	KLOC + cost drivers + **phase-wise effort distribution**.	Most accurate

Note:

- **Basic** → Simple, only size-based.
- **Intermediate** → Size + cost drivers.
- **Detailed** → Size + cost drivers + phase/module analysis.

Software Engineering (303105253)

92. What is project scheduling?
93. Why is scheduling important in software projects?
94. Define staffing in the context of software projects.
95. Explain the relationship between **effort, schedule, and staffing**.
96. What is risk in software project management?
97. Define **risk identification**. Give examples of **technical**, **business**, and **schedule** risks
98. What is risk assessment? Explain **risk probability and impact matrix**
99. Explain **risk mitigation**, **risk monitoring**, and **risk contingency planning**.
100. What are the common risk mitigation strategies?
101. Differentiate between **risk avoidance** and **risk acceptance**.
102. What is **Earned Value Analysis (EVA)** in project monitoring?
103. Explain the role of **elicitation, analysis, specification, validation, and management** in RE.
104. Define Termonology and its types.
105. What is a Software Requirement Specification (SRS)?
106. What is a use case in software engineering?
107. Explain the structure of a use case (Actor, Precondition, Steps, Postcondition).
108. Differentiate between **primary actor** and **secondary actor** in a use case.
109. What is the difference between a **use case diagram** and a **use case description**?
110. Give an example of a use case for an ATM system.
111. Explain the role of **stakeholders** in validation.
112. Differentiate between **verification** and **validation** of requirements.
113. Differentiate between requirement analysis and requirement elicitation.
114. What is the purpose of an RMMM plan in software project management?
115. What are software metrics? Why are they important? Differentiate between **process metrics, project metrics, and product metrics** with examples