SEN-2025-May-PYQ

Q1. [20 Marks]

a. Explain CMM Model in software development.

b. Explain the user interface design principle.

c. Explain 3 P’s in software project spectrum.

d. Explain importance of WBS in software engineering with example.

Q2. [10 Marks]

a. Explain about different types of software prototypes? Discuss prototype model in detail.

b. Explain the characteristics of SRS? Also discuss general format of software requirement specification.

Q3. [10 Marks]

a. Explain analysis model elements? Draw class diagram and swim lane diagram for online food ordering system.

b. Explain the design principles and concepts? Also explain difference between coupling and cohesion

Q4. [10 Marks]

a. Explain about COCOMO II model with example.

b. Explain the steps of software quality assurance plan.

Q5. [10 Marks]

a. Explain about CPM and PERT project scheduling technique in detail.

b. Explain steps of change control process in SCM in detail?

Q6. [10 Marks]

a. Write a short note on Software Re-engineering and different types of software maintenance?

b. Write a note on Alpha Beta and White Box testing?

Q1. [20 Marks] - Answers

a. Explain CMM Model in software development.

CMM Model (Capability Maturity Model)

- **Definition:**

  The CMM (Capability Maturity Model) is a framework that helps organizations improve their software development processes in a structured and gradual manner.

- **Purpose:**

  It measures the maturity of an organization's software process and guides improvements for higher quality and efficiency.

- **Developed by:**

  Software Engineering Institute (SEI), Carnegie Mellon University.

Five Maturity Levels:

1. **Level 1 – Initial:**

   Process is ad-hoc and unpredictable; success depends on individual effort.

2. **Level 2 – Repeatable:**

   Basic project management processes are established; similar projects can be repeated.

3. **Level 3 – Defined:**

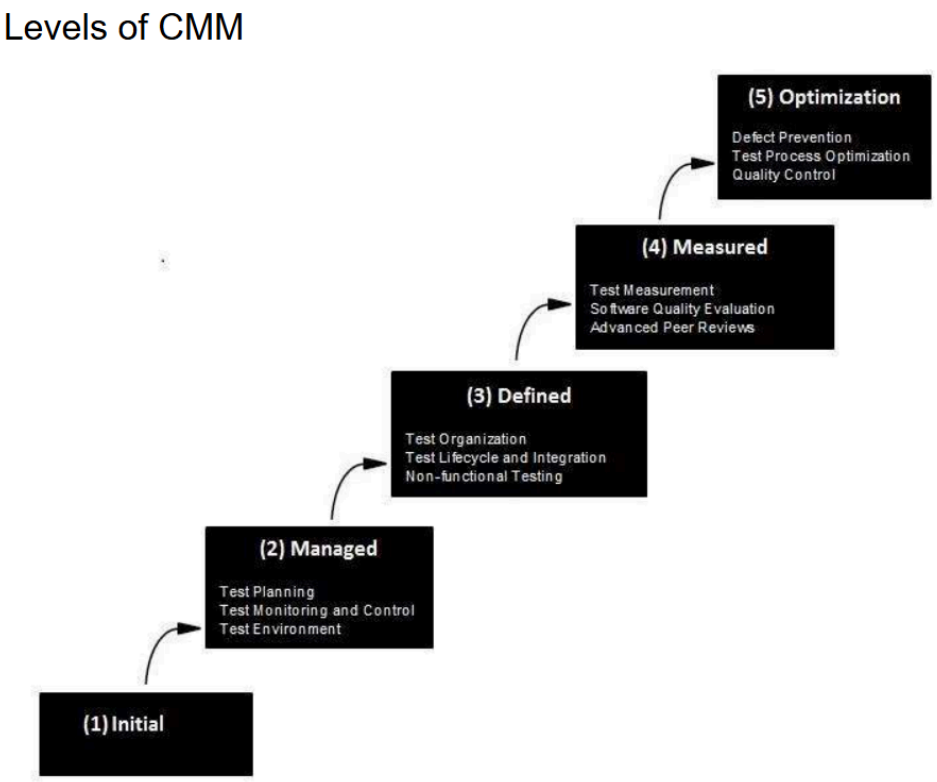
Processes are documented, standardized, and integrated into a standard process for the organization.

4. **Level 4 – Managed:**

   Processes are measured and controlled using quantitative techniques.

5. **Level 5 – Optimizing:**

   Continuous process improvement through feedback and innovation.



b. Explain the user interface design principle.

User Interface (UI) Design Principles

User Interface Design Principles guide developers in creating interfaces that are **easy to use, efficient, and user-friendly**.

Key Principles:

1. **Clarity:**

   Interface should clearly communicate information — users must understand what actions are possible and what each element does.

2. **Consistency:**

   Use uniform colors, fonts, buttons, and layouts across the application for predictable user experience.

3. **Feedback:**

   System should provide responses (e.g., messages, progress bars) to inform users that their actions have been received or completed.

4. **Simplicity:**

   Keep design simple and intuitive — avoid unnecessary elements or complex navigation.

5. **Flexibility and Efficiency:**

   Allow shortcuts or customization for experienced users while keeping the interface simple for beginners.

c. Explain 3 P’s in software project spectrum.

3 P’s in Software Project Spectrum

The **3 P’s** represent the key dimensions that determine the success of any software project: **People, Product, and Process.**

1. People:

- Refers to everyone involved in the project — developers, testers, managers, and clients.
- Skilled, motivated, and well-coordinated team members are essential.
- Good communication and teamwork improve productivity and quality.

2. Product:

- Defines **what** is to be built — the software's goals, features, and constraints.
- Understanding user needs and system requirements ensures the right product is developed.
- Clear product vision helps guide design and implementation.

3. Process:

- Refers to **how** the software is developed — the methods, models, and tools used.
- A well-defined process (like Agile or Waterfall) ensures systematic development, testing, and delivery.
- Helps maintain quality, reduce risk, and manage time and cost effectively.

d. Explain importance of WBS in software engineering with example.

Importance of WBS (Work Breakdown Structure) in Software Engineering

Definition:

A **Work Breakdown Structure (WBS)** is a hierarchical decomposition of the total project work into smaller, manageable tasks or modules.

Importance:

1. **Clear Project Structure:**

   Breaks complex projects into smaller, understandable components.

2. **Better Planning and Scheduling:**

   Helps estimate time, cost, and resources for each task accurately.

3. **Improved Accountability:**

Assigns clear responsibilities to team members for specific tasks.

4. **Progress Tracking:**

   Makes it easier to monitor and control project progress.

5. **Risk Management:**

   Identifies critical areas early, reducing chances of failure or delay.

6. **Enhanced Communication:**

   Provides a clear roadmap for the team and stakeholders.

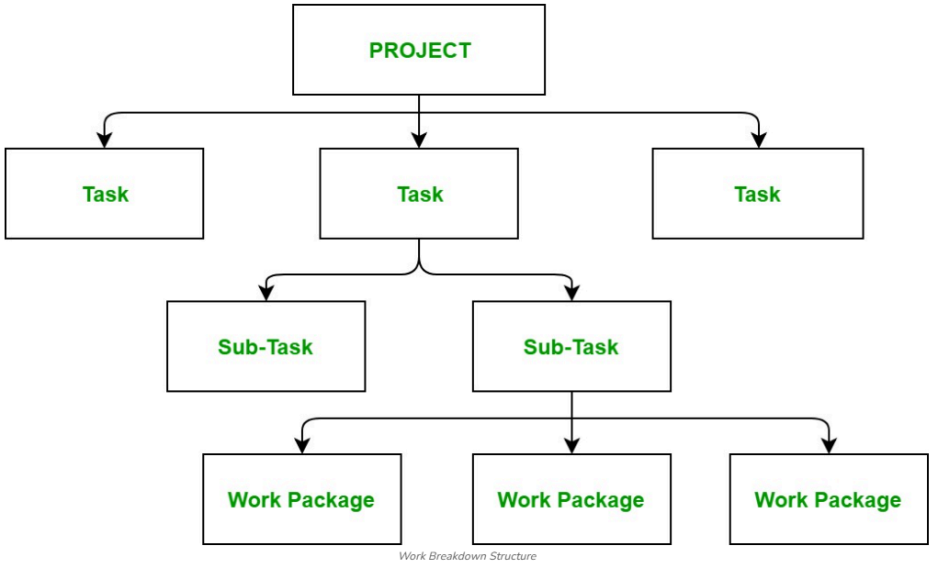
Example:

Project: Online Shopping System

WBS Breakdown:

1. **Requirement Analysis**
2. **System Design**
   - UI Design
   - Database Design
3. **Development**
   - Frontend Coding
   - Backend Coding
4. **Testing**
   - Unit Testing
   - Integration Testing
5. **Deployment & Maintenance**

A **Work Breakdown Structure (WBS) diagram:**



Work Breakdown Structure

Q2. [10 Marks] - Answers

a. Explain about different types of software prototypes? Discuss prototype model in detail.

Types of Software Prototypes

Software prototypes are early versions of a system used to visualize features, gather feedback, and refine requirements. Common types include:

1. **Throwaway (Rapid) Prototype**

- Built quickly to understand user requirements.
- Discarded after use; not part of final system.
- Useful for refining unclear or incomplete requirements.

2. **Evolutionary Prototype**

- Developed and improved iteratively.
- Becomes the core of the final system.
- Suitable for complex systems with evolving requirements.

3. **Incremental Prototype**

- System is built in parts (increments).
- Each prototype adds functionality.
- Final system is the sum of all increments.

4. **Extreme Prototyping (for Web Apps)**

- Three phases: static HTML screens → functional data services → integration.
- Common in web development for fast UI feedback.

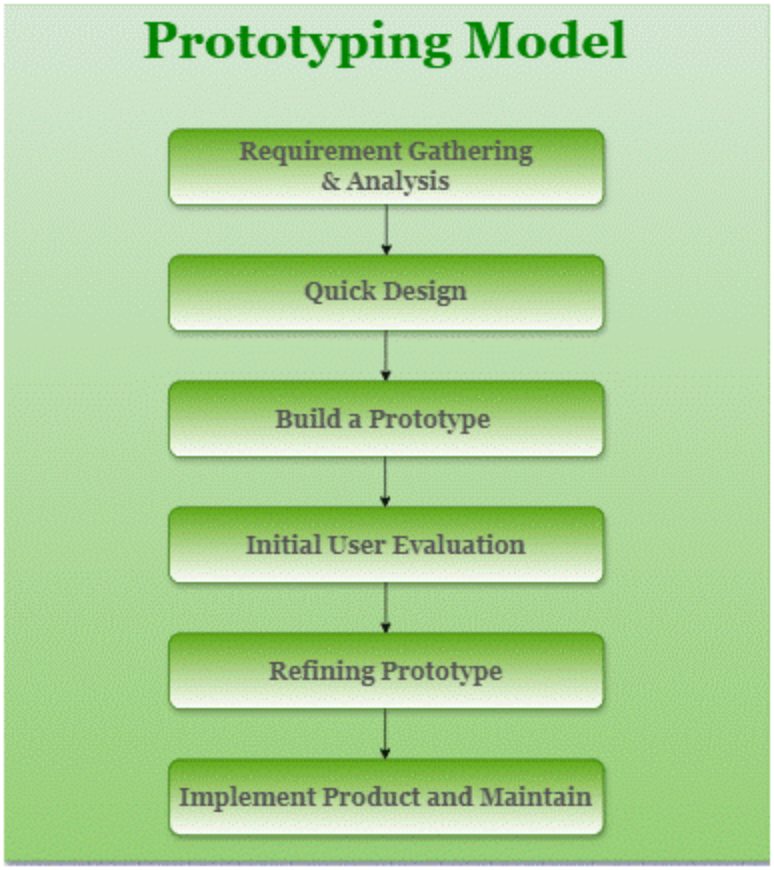
Prototype Model (Process Model)

The Prototype Model is a software development approach focused on building a working prototype before final development.

Steps in the Prototype Model

1. **Requirement Gathering**
    - Initial requirements are collected from users.
2. **Quick Design**
    - A basic design is created focusing on visible aspects (UI, workflow).
3. **Prototype Building**
    - A working model is developed with limited functionality.
4. **User Evaluation**
    - Users test the prototype and give feedback.
5. **Refinement**
    - Requirements and design are updated based on feedback.
6. **Final Product Development**

- Once requirements are clear, the actual system is built using a suitable model (e.g., Waterfall).



Prototyping Model Phases

b. Explain the characteristics of SRS? Also discuss general format of software requirement specification.

Characteristics of Software Requirement Specification (SRS)

SRS defines what the software will do and how it will interact with users and systems. Key characteristics include:

1. **Correctness**

- Every requirement stated must be accurate and reflect the actual needs of stakeholders.

2. **Completeness**

- All functional and non-functional requirements must be included.
- No missing scenarios or undefined behaviors.

3. **Unambiguity**

- Requirements should be stated clearly without multiple interpretations.

4. **Consistency**

- No conflicting requirements.
- Terminology and logic should remain uniform throughout the document.

5. **Verifiability**

- Each requirement must be testable through inspection, analysis, or testing.

6. **Modifiability**

- Easy to update and maintain as requirements evolve.
- Structured format helps in tracking changes.

7. **Traceability**

- Each requirement should be traceable to its origin (stakeholder, use case).
- Helps in impact analysis and validation.

General Format of SRS Document

A typical SRS document follows IEEE 830 standard and includes:

1. **Introduction**

- Purpose of the system
- Scope

- Definitions, acronyms, abbreviations
- References
- Overview of the document

2. **Overall Description**

- Product perspective (existing systems, interfaces)
- Product functions (high-level features)
- User characteristics
- Constraints (hardware, software, legal)
- Assumptions and dependencies

3. **Specific Requirements**

- Functional requirements (detailed features and operations)
- Non-functional requirements (performance, security, usability)
- External interface requirements (UI, hardware, software, communication)

4. **Appendices**

- Supporting information, diagrams, glossary

5. **Index**

- For easy navigation of the document

Q3. [10 Marks] - Answers

a. Explain analysis model elements? Draw class diagram and swim lane diagram for online food ordering system.

Analysis Model Elements

Definition:

The **Analysis Model** represents **what** of the system — it defines what the system must do based on user requirements before moving to design.

Key Elements of the Analysis Model:

1. **Scenario-Based Elements:**
   - Describe how users interact with the system.
   - Includes **Use Case Diagrams** and **Use Case Descriptions**.
   - Example: "User logs in to view order history."

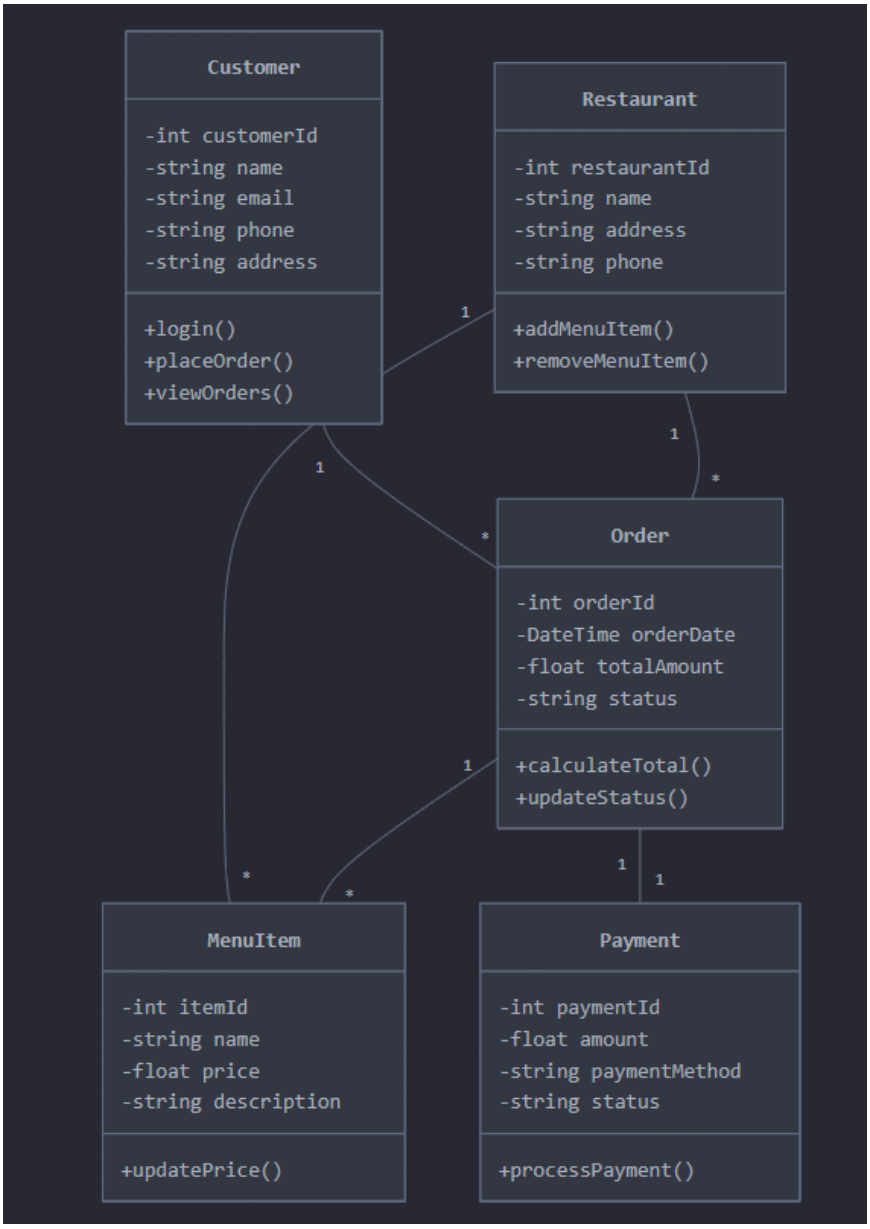
2. **Class-Based Elements:**
   - Identify **objects (classes)** in the system and their relationships.
   - Includes attributes, operations, and associations.
   - Example: Classes like *Customer*, *Order*, *Product*.

3. **Behavioral Elements:**
   - Show **dynamic behavior** of the system.
   - Represented using **State Diagrams** or **Sequence Diagrams**.
   - Example: How an order changes state from "Placed" → "Shipped" → "Delivered."

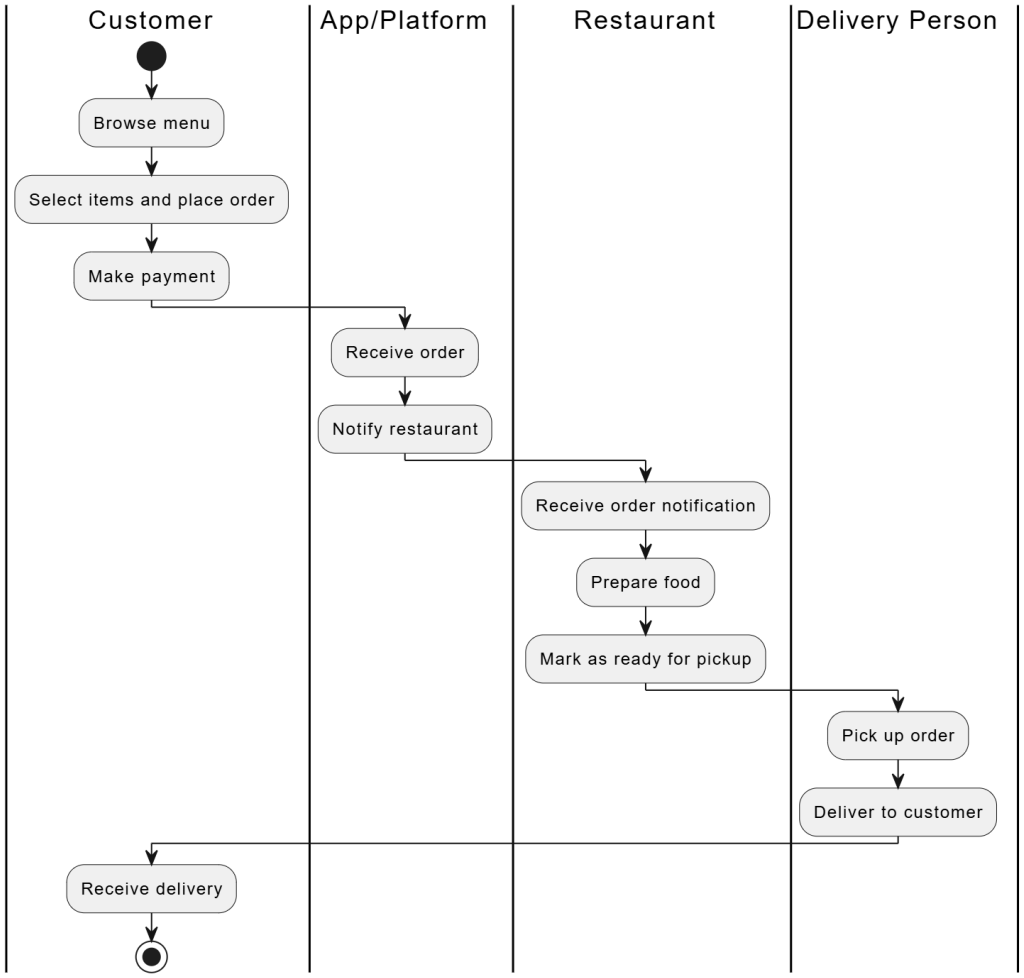
4. **Flow-Oriented Elements:**
   - Represent the **flow of data and transformations** in the system.
   - Includes **Data Flow Diagrams (DFDs)**.
   - Example: Data moves from *User Input* → *Validation* → *Database*.

5. **Functional Elements:**
    - Define **functions or operations** the system performs.
    - Derived from use cases and functional requirements.
    - Example: Login validation, order processing, payment calculation.

Class diagram:



Swim Lane diagram:



b. Explain the design principles and concepts? Also explain difference between coupling and cohesion

Software Design Principles and Concepts

Design principles guide developers in creating software that is maintainable, scalable, and efficient. Key principles include:

1. **Modularity**

- Break the system into smaller, manageable modules.
- Each module performs a specific function.

2. **Abstraction**

- Hide internal details and expose only essential features.
- Simplifies complexity and enhances reusability.

3. **Encapsulation**

- Bind data and functions together.
- Protects internal state and promotes security.

4. **Separation of Concerns**

- Divide responsibilities across components (e.g., UI, logic, data).
- Improves clarity and maintainability.

5. **DRY (Don’t Repeat Yourself)**

- Avoid code duplication.
- Promotes reuse and reduces errors.

6. **Single Responsibility Principle**

- Each module/class should have one clear purpose.
- Makes code easier to test and modify.

7. **Open/Closed Principle**

- Software entities should be open for extension but closed for modification.

- Encourages flexible and robust design.

Coupling vs Cohesion

These are key concepts in modular design:

Aspect	Coupling	Cohesion
Definition	Degree of interdependence between modules	Degree to which elements within a module belong together
Goal	Minimize	Maximize
Ideal State	Low Coupling	High Cohesion
Impact	Low coupling improves flexibility and reusability	High cohesion improves clarity and maintainability
Example	Module A directly accessing Module B's data → high coupling	Module handling all user input tasks → high cohesion

Q4. [10 Marks] - Answers

a. Explain about COCOMO II model with example.

COCOMO II Model (Constructive Cost Model II)

---

1. Introduction:

- **COCOMO II** is an advanced version of the original **COCOMO (Constructive Cost Model)** developed by **Barry Boehm**.
- It is used to **estimate the cost, effort, and schedule** required to develop a software project.
- The model considers modern software development practices like **reuse, prototyping, and object-oriented development**.

---

2. Purpose:

- To predict the **effort (in person-months)**, **development time**, and **cost** of a software project based on its size and complexity.

- Helps project managers plan resources and schedules accurately.

3. Sub-Models of COCOMO II:

1. **Application Composition Model:**

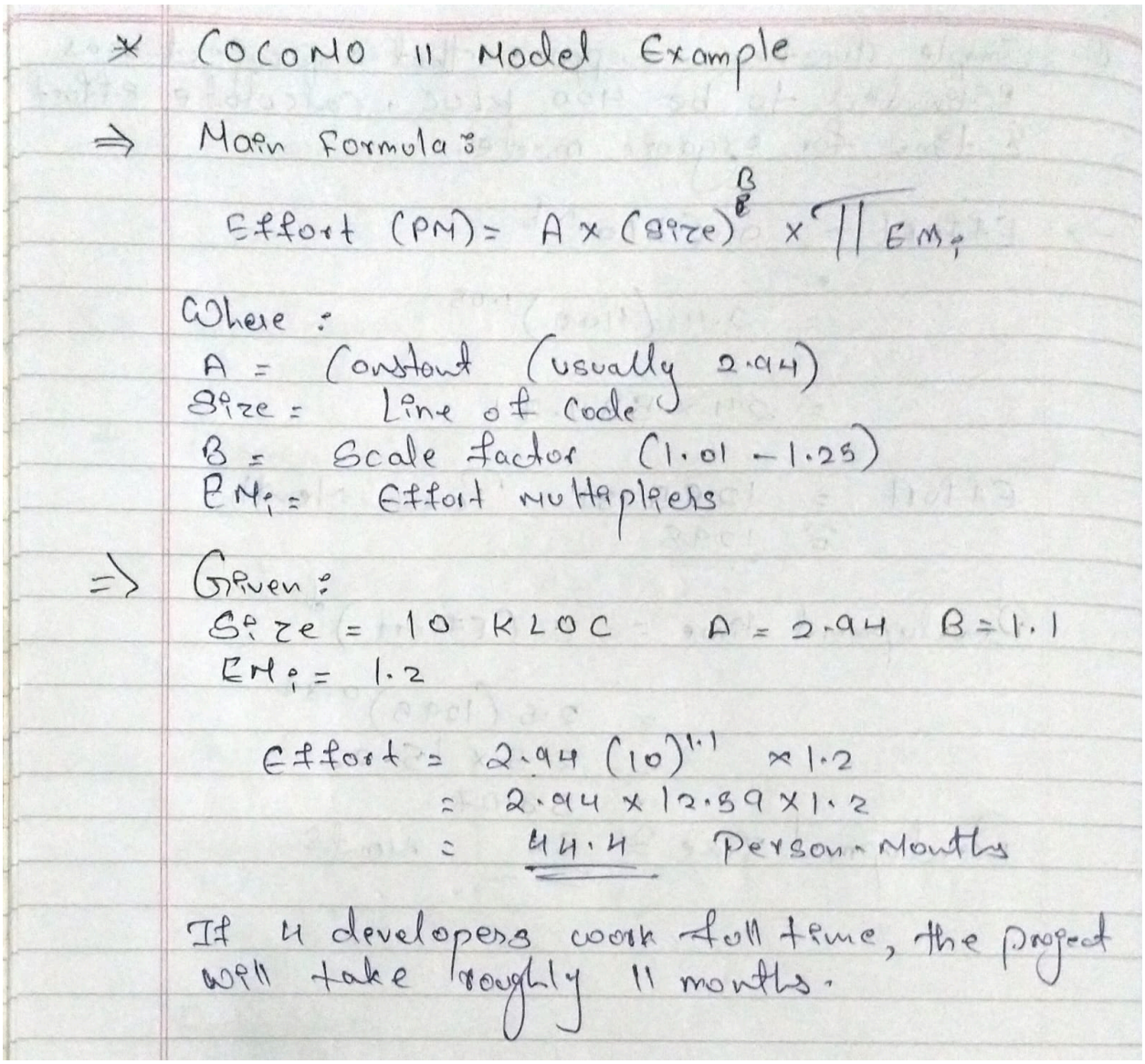
   Used in the early prototyping stage, based on **Object Points (OP)**.

2. **Early Design Model:**

   Used when overall architecture is known; based on **Unadjusted Function Points (UFP)** and approximate effort multipliers.

3. **Post-Architecture Model:**

   Used when detailed design is available; considers **17 cost drivers** and **5 scale factors**.



b. Explain the steps of software quality assurance plan.

Software Quality Assurance (SQA) Plan – Steps

A Software Quality Assurance Plan outlines the procedures, standards, and activities to ensure software meets quality requirements. The key steps include:

1. **Purpose and Scope Definition**

- Define the goals of the SQA plan.
- Specify what software/system is covered and the quality objectives.

2. **Reference Documents**

- List all related documents like project plans, standards, and policies.
- Ensures consistency and traceability.

3. **Software Quality Objectives**

- Define measurable quality goals (e.g., defect density, performance benchmarks).
- Align with customer expectations and project constraints.

4. **SQA Responsibilities**

- Assign roles and responsibilities to QA team members.
- Clarify who performs reviews, audits, testing, and reporting.

5. **Standards, Practices, and Conventions**

- Specify coding standards, documentation guidelines, and development practices.
- Ensure uniformity and compliance across the project.

6. **SQA Activities**

- Include reviews, audits, walkthroughs, testing, and defect tracking.
- Define when and how each activity will be performed.

7. **Tools, Techniques, and Methodologies**

- List tools used for testing, version control, static analysis, etc.
- Mention methodologies like Agile, Waterfall, or DevOps if applicable.

8. **Reviews and Audits**

- Plan for formal technical reviews, code inspections, and process audits.
- Helps detect issues early in the lifecycle.

9. **Test Plan and Strategy**

- Outline test levels (unit, integration, system, acceptance).
- Define test environments, data, and success criteria.

10. **Problem Reporting and Corrective Action**

- Describe how defects will be reported, tracked, and resolved.
- Include escalation procedures and re-testing steps.

11. **Records and Metrics**

- Specify what quality records will be maintained (e.g., test results, review logs).
- Define metrics to monitor quality trends and process effectiveness.

12. **SQA Reporting and Feedback**

- Define reporting structure and frequency (e.g., weekly QA reports).
- Ensure feedback loops to improve processes continuously.

Q5. [10 Marks] - Answers

a. Explain about CPM and PERT project scheduling technique in detail.

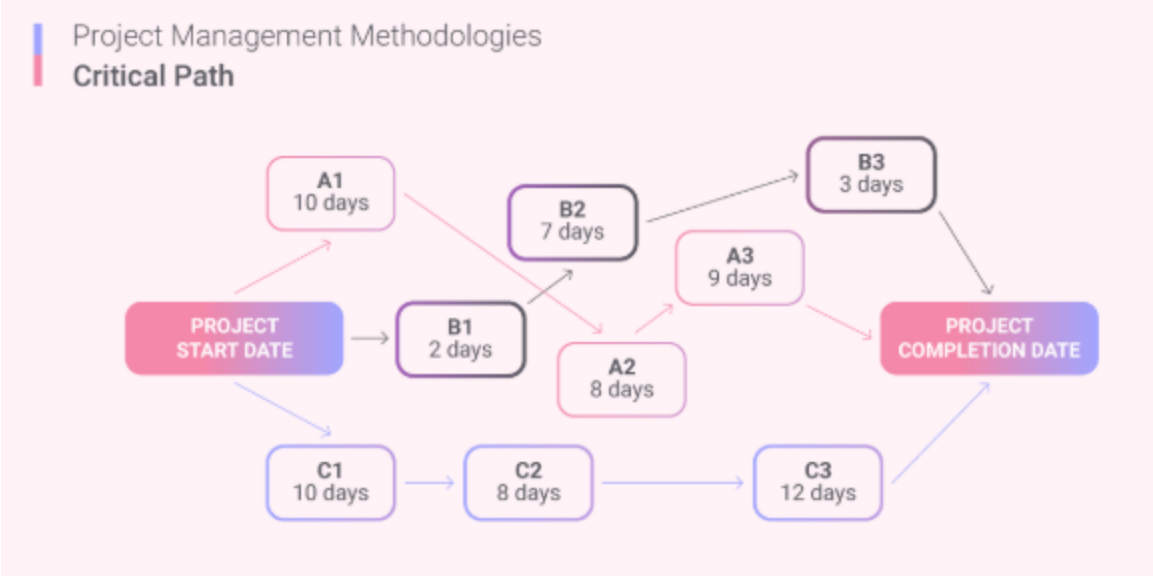
1. Critical Path Method (CPM)

Definition

CPM is a deterministic technique used to identify the longest path of planned activities to the end of the project — known as the *critical path*. It helps determine the shortest possible project duration.

Key Concepts

- **Activities**: Tasks required to complete the project.
- **Dependencies**: Logical order of activities.
- **Critical Path**: Longest path through the network; determines minimum project time.
- **Slack/Float**: Time an activity can be delayed without affecting the project timeline.



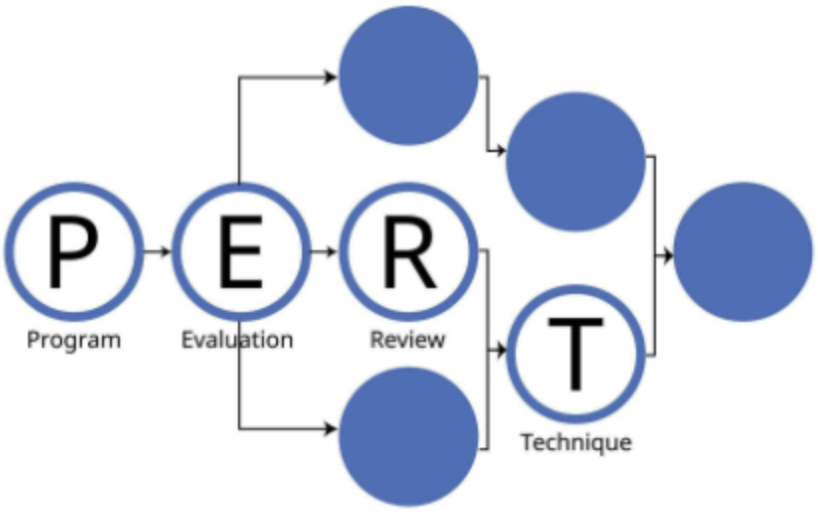
Use Case

- Suitable for projects with **predictable activity durations** (e.g., construction, manufacturing).
- Helps in **resource allocation** and **deadline tracking**.

2. Program Evaluation and Review Technique (PERT)

Definition

PERT is a probabilistic technique used to estimate project duration when activity times are uncertain. It uses three time estimates to calculate expected duration.



Time Estimates

- **Optimistic (O)**: Minimum time required.
- **Most Likely (M)**: Best guess of time.
- **Pessimistic (P)**: Maximum time required.

Expected Time Formula

$$TE = \frac{O + 4M + P}{6}$$

Use Case

- Ideal for **R&D, software development**, or any project with **uncertain timelines**.
- Helps in **risk analysis** and **schedule flexibility**.

b. Explain steps of change control process in SCM in detail?

Change Control Process in Software Configuration Management (SCM)

Change control ensures that all modifications to software artifacts are systematically evaluated, approved, and tracked to maintain integrity and traceability.

1. **Identification of Change**

- A change request is raised due to bug fixes, enhancements, or requirement shifts.
- Includes description, reason, and affected components.

2. **Change Request Submission**

- Formal change request (CR) is submitted to the Change Control Board (CCB).
- Logged into a change management system with a unique ID.

3. **Impact Analysis**

- Technical and business teams assess the impact on cost, schedule, resources, and quality.
- Identifies affected modules, dependencies, and risks.

4. **Change Evaluation and Approval**

- CCB reviews the analysis and decides to approve, reject, or defer the change.
- Decision is documented with justification.

5. **Implementation of Change**

- Approved changes are assigned to developers.
- Code, documents, and test cases are updated accordingly.

6. **Verification and Validation**

- QA team tests the change to ensure correctness and no side effects.
- Regression testing may be performed.

7. **Update Baselines**

- Once validated, the updated configuration item becomes part of the new baseline.
- Ensures version control and traceability.

8. **Documentation and Communication**

- All changes are documented in change logs.
- Stakeholders are informed about the change status and impact.

9. **Audit and Review**

- Periodic audits ensure compliance with SCM policies.
- Helps in continuous improvement of the change control process.

Q6. [10 Marks] - Answers

a. Write a short note on Software Re-engineering and different types of software maintenance?

Software Re-engineering

Software Re-engineering is the process of analyzing and modifying existing software to improve its functionality, performance, maintainability, or adaptability without changing its core purpose.

Key Activities

- **Reverse Engineering**: Understanding existing code and design.
- **Restructuring**: Improving code structure, logic, or documentation.
- **Refactoring**: Enhancing internal code quality without altering behavior.
- **Forward Engineering**: Rebuilding the system using modern technologies.

Benefits

- Extends software life.
- Reduces maintenance cost.
- Improves performance and scalability.

Types of Software Maintenance

Software maintenance involves updating software after delivery to correct faults or improve performance.

1. **Corrective Maintenance**

- Fixes bugs and defects.
- Example: Resolving runtime errors or logic flaws.

2. **Adaptive Maintenance**

- Modifies software to work in new environments.
- Example: Updating for new OS or hardware compatibility.

3. **Perfective Maintenance**

- Enhances performance or adds new features.
- Example: Improving UI or optimizing database queries.

4. **Preventive Maintenance**

- Anticipates future issues and improves reliability.
- Example: Code cleanup, updating libraries to avoid future failures.

b. Write a note on Alpha, Beta and White Box testing?

Alpha Testing

Definition

Alpha testing is performed by internal teams (developers or QA) before releasing the software to external users.

Key Points

- Conducted in a controlled environment.
- Focuses on identifying bugs and usability issues.
- Involves both **white-box** and **black-box** techniques.
- Feedback is used to refine the product before beta release.

Beta Testing

Definition

Beta testing is done by real users in a real environment to validate the product under actual usage conditions.

Key Points

- Conducted after alpha testing.
- Helps uncover unexpected issues and gather user feedback.
- Often used to assess **user satisfaction**, **performance**, and **reliability**.
- May be open (public) or closed (limited group).

White Box Testing

Definition

White box testing (also called structural or glass-box testing) involves testing the internal logic and structure of the code.

Key Points

- Testers need knowledge of the source code.
- Focuses on paths, conditions, loops, and branches.
- Common techniques: **statement coverage**, **branch coverage**, **path coverage**.
- Helps ensure code correctness and optimization.