SEN-2024-December-PYQ

Q1. [20 Marks]

a. Describe the characteristics, nature of software and explain the layered structure of software engineering?

b. Explain prototyping concept required in Spiral software process model?

c. Explain about Object based estimation technique?

d. Explain the golden rules of user interface design?

e. Explain about software Re-engineering and reverse engineering?

Q2. [10 Marks]

a. Explain scrum agile development model and scrum process in detail?

b. Explain what is a V-Model? And discuss any one type of incremental process model?

Q3. [10 Marks]

a. Explain the design model and draw deployment diagram, swim lane diagram for online shopping?

b. Explain about refactoring, cohesion and coupling also benefits of high cohesion and low coupling?

Q4. [10 Marks]

a. Explain Earned Value Analysis? Find project EAC, ETC, where AC is 15000, BAC is 22000, EV is 13000, CPI is 0.8?

Q5. [10 Marks]

a. Explain about Project scheduling and anyone tracking Technique with example?

Q6. [10 Marks]

a. Explain the principles of software testing? Discuss in detail the difference between black box and white box testing technique?

Q1. [20 Marks] - Answers

a. Describe the characteristics, nature of software and explain the layered structure of software engineering?

Characteristics and Nature of Software

1. Characteristics of Software:

- **Intangible:** Software cannot be seen or touched; it only operates logically.
- **Engineered Product:** Software is developed, not manufactured — it requires design and creativity.
- **Doesn't Wear Out:** Unlike hardware, software doesn't degrade physically but can fail due to bugs or changes.
- **Custom-Built:** Each software product is unique, developed for specific user needs.
- **Evolves Over Time:** Software must be maintained and updated to adapt to new requirements or environments.

2. Nature of Software:

- **Logical rather than physical:** Exists as code, not a tangible object.
- **Complex and abstract:** Involves logic, data structures, and algorithms.
- **Dependent on human intelligence:** Quality depends on developer skill and design accuracy.

Layered Structure of Software Engineering

Software engineering follows a **layered technology approach**, where each layer supports the one above it.

Layers:

1. **Quality Focus (Top Layer):**
    - Ensures that every activity aims for **high-quality software**.
    - Quality is the foundation for customer satisfaction and reliability.
2. **Process Layer:**
    - Defines the **framework** for software development (e.g., Agile, Waterfall).
    - Guides planning, control, and coordination of all activities.
3. **Methods Layer:**
    - Provides **technical methods** for analysis, design, coding, testing, and maintenance.
4. **Tools Layer (Bottom Layer):**
    - Provides **automated support** for methods and processes (e.g., IDEs, testing tools, version control).

b. Explain prototyping concept required in Spiral software process model?

Prototyping in the Spiral Model

Definition:

In the **Spiral Model**, **prototyping** is used to **understand and refine requirements** by developing an early version of the system.

It helps both developers and users visualize the product before full-scale development.

Key Concepts:

1. **Risk Reduction Technique:**
    - Prototyping is used in each spiral cycle to **identify and reduce risks** related to unclear requirements or design issues.
2. **User Involvement:**
    - Users review the prototype and provide feedback, ensuring that the final system aligns with their expectations.
3. **Progressive Refinement:**
    - The prototype is **revised and improved** in successive spirals until it evolves into the complete system.
4. **Early Validation:**
    - Enables early detection of design or requirement errors, reducing costly rework later.
5. **Decision Support:**
    - After evaluating the prototype, stakeholders decide whether to **proceed, modify, or stop** the project.

c. Explain about Object based estimation technique?

Object-Based Estimation Technique

Definition:

The **Object-Based Estimation Technique** is a **software cost estimation method** used mainly in **object-oriented development**.

It estimates the **effort, time, and cost** based on the **number and complexity of objects** in the system.

Key Concepts:

1. **Based on Object Classes:**

- The system is divided into **classes and objects** (e.g., GUI objects, database objects, control objects).
- Each type of object has an **assigned effort value** based on its complexity.

2. **Effort Estimation:**
   - The total effort is calculated by **summing up the effort for all identified objects**.
   - Example: Effort = Σ (Number of Objects × Effort per Object Type)

3. **Reusability Consideration:**
   - Since object-oriented systems often reuse classes, this technique **reduces effort** for reused components.

4. **Supports Early Estimation:**
   - Can be applied **during design**, even before full coding begins.

5. **Useful for Object-Oriented Projects:**
   - Ideal for systems developed in **languages like Java, C++, or Python**, where software is structured around objects.

Example:

Suppose a project has:

- 10 simple objects × 5 person-hours = 50
- 5 medium objects × 10 person-hours = 50
- 2 complex objects × 20 person-hours = 40

  Total Effort = 140 person-hours

d. Explain the golden rules of user interface design?

Golden Rules of User Interface (UI) Design

Proposed by **Ben Shneiderman**, the **Eight Golden Rules** provide guidelines to design **effective, user-friendly, and consistent interfaces**.

Key Rules:

1. **Strive for Consistency:**
   - Use consistent terminology, colors, layouts, and actions throughout the interface.
   - Example: Same button style for “Submit” on all pages.

2. **Enable Frequent Users to Use Shortcuts:**
   - Provide accelerators like **keyboard shortcuts** or **auto-completion** for expert users to save time.

3. **Offer Informative Feedback:**
   - The system should respond clearly to every user action (e.g., progress bars, confirmation messages).

4. **Design Dialogs to Yield Closure:**
   - Group related actions into clear sequences with a **beginning, middle, and end**.
   - Example: “Order Placed Successfully” message after payment.

5. **Offer Simple Error Handling:**
   - Prevent errors where possible; provide **clear, helpful messages** when errors occur.

6. **Permit Easy Reversal of Actions:**
    - Allow users to **undo or redo** operations easily, reducing fear of making mistakes.

7. **Support Internal Locus of Control:**
    - Users should feel **in control**, not controlled by the system. Avoid unexpected actions or pop-ups.

8. **Reduce Short-Term Memory Load:**
    - Keep screens simple, avoid requiring users to remember data from one screen to another.

e. Explain about software Re-engineering and reverse engineering?

1. Software Re-engineering:

Definition:

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

Key Points:

- Involves **restructuring, cleaning, and updating** old code.
- Helps **extend the life** of legacy systems.
- May include activities like **code refactoring, documentation update, and migration** to new platforms.

- Goal: **Enhance software quality and reduce future maintenance costs.**

Example:

Converting an old **C-based desktop application** into a **modern Java web application**.

2. Reverse Engineering:

Definition:

Reverse Engineering is the process of **analyzing existing software to understand its design, structure, and functionality** when documentation is missing or outdated.

Key Points:

- Used to **recover design details** or **generate documentation** from source code.
- Helps in **understanding legacy systems** or integrating with new systems.
- Often a **first step in re-engineering**.

Example:

Extracting UML diagrams from source code to understand system flow.

Q2. [10 Marks] - Answers

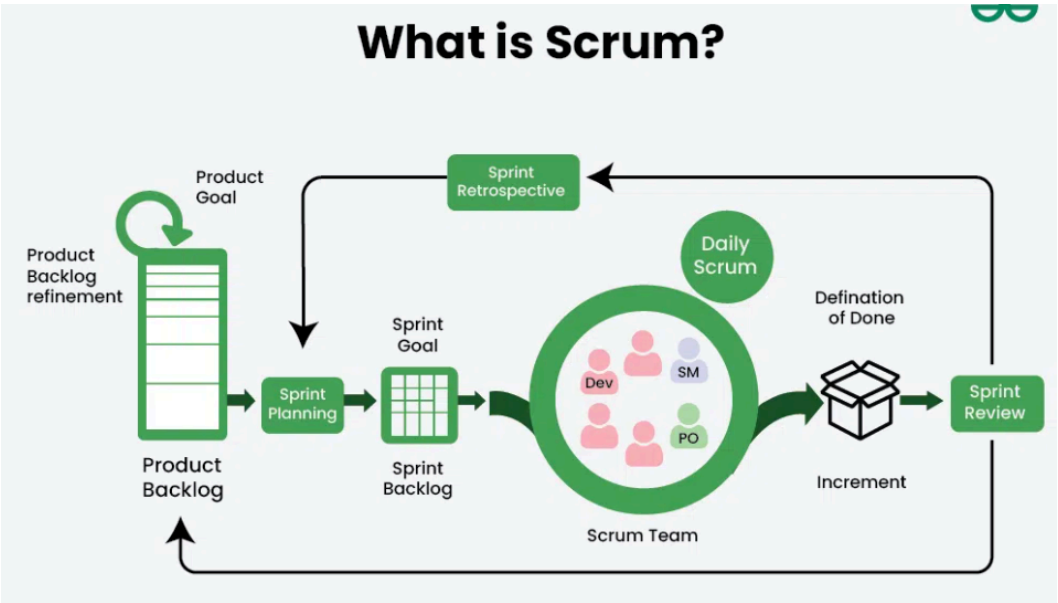
a. Explain scrum agile development model and scrum process in detail?

What Is Scrum?

Scrum is a lightweight, flexible framework used in Agile project management. It helps teams deliver value incrementally by breaking work into manageable units

and continuously improving through feedback and reflection.

- **Origin**: Introduced in the early 1990s.
- **Focus**: Transparency, inspection, and adaptation.
- **Team Structure**: Cross-functional and self-organizing.



Scrum Roles

Role	Responsibility
Product Owner	Defines product vision, manages backlog, prioritizes features.
Scrum Master	Facilitates Scrum process, removes blockers, ensures team follows Agile principles.
Development Team	Builds the product, estimates tasks, commits to sprint goals.

Scrum Process Overview

1. **Product Backlog**

- A prioritized list of features, enhancements, and fixes.

- Managed by the Product Owner.

2. **Sprint Planning**

- Team selects items from the product backlog to work on during the sprint.
- Defines **Sprint Goal** and **Sprint Backlog**.

3. **Sprint (1–4 weeks)**

- Time-boxed development cycle.
- Team works on selected tasks collaboratively.

4. **Daily Scrum (Stand-up)**

- 15-minute daily meeting.
- Team members share progress, plans, and blockers.

5. **Sprint Review**

- Held at the end of the sprint.
- Team demonstrates completed work to stakeholders.
- Feedback is collected for future improvements.

6. **Sprint Retrospective**

- Team reflects on the sprint process.
- Identifies what went well, what didn’t, and how to improve.

7. **Increment**

- A potentially shippable product delivered at the end of each sprint.

Scrum Artifacts

- **Product Backlog**: Master list of all desired work.
- **Sprint Backlog**: Subset of product backlog selected for the sprint.
- **Increment**: The completed, usable product output.

b. Explain what is a V-Model? And discuss any one type of incremental process model?

V-Model answer is in the 2023 May Answer sheet

Incremental Process Model – Example: Staged Delivery Model

The **Incremental Model** develops software in small, manageable parts (increments). Each increment adds functionality and is tested independently.

Staged Delivery Model

- Delivers the system in predefined stages.
- Each stage includes analysis, design, coding, and testing.
- Early increments provide core features; later ones add enhancements.

Benefits

- Early delivery of working software.
- Easier to test and debug smaller modules.
- Flexible to changing requirements.

Limitations

- Requires careful planning of increments.

- Integration can become complex over time.

Q3. [10 Marks] - Answers

a. Explain the design model and draw deployment diagram, swim lane diagram for online shopping?

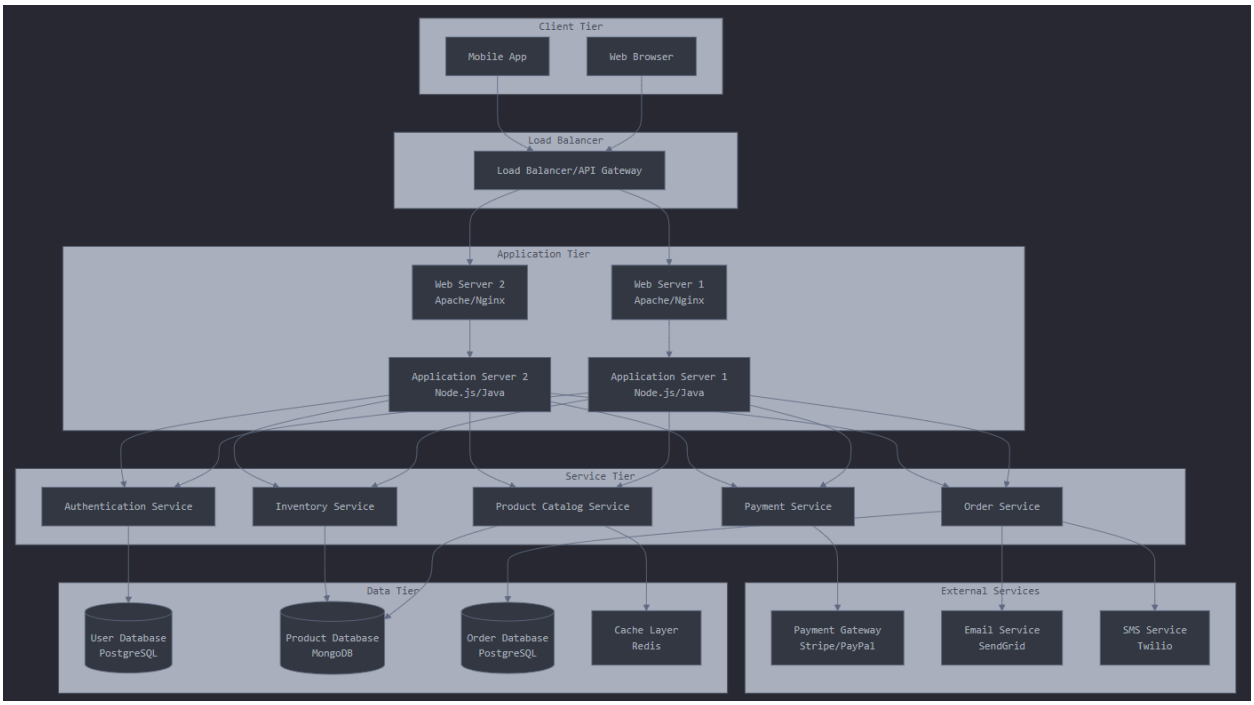
What Is a Design Model?

A **design model** in software engineering is a set of representations that describe the architecture, components, interfaces, and data flow of a system. It bridges the gap between requirements and implementation.

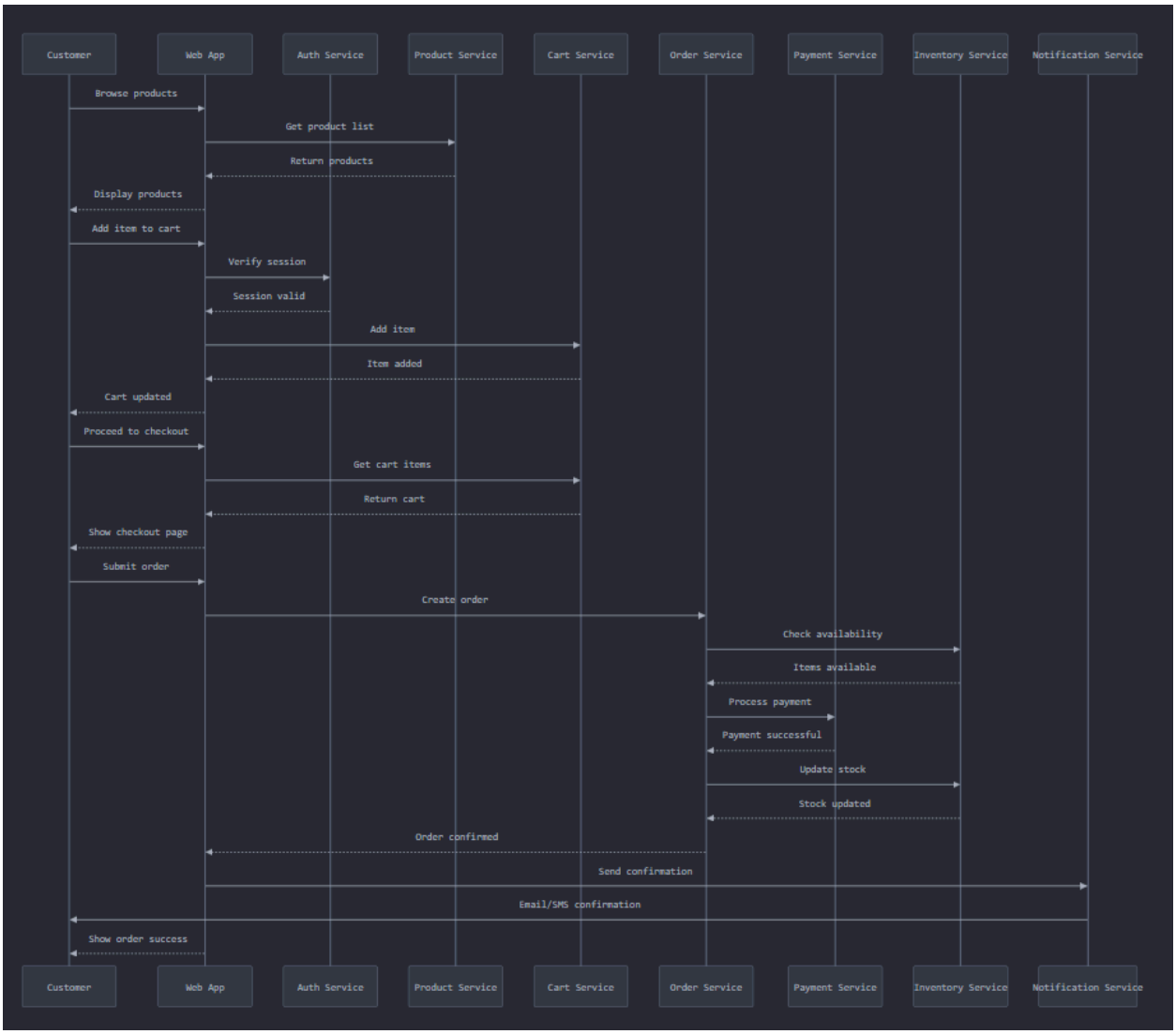
Key Elements of a Design Model

- **Architectural Design**: Defines the overall structure and relationships among major components.
- **Component Design**: Specifies internal logic of each module.
- **Interface Design**: Describes how components interact with each other and with users.
- **Data Design**: Models how data is stored, accessed, and manipulated.
- **Deployment Design**: Shows how software components are distributed across hardware.

Deployment Diagram:



Swim Lane Diagram:



b. Explain about refactoring, cohesion and coupling also benefits of high cohesion and low coupling?

Refactoring

Refactoring is the process of improving the internal structure of existing code without changing its external behavior.

Key Points:

- Focuses on **code readability**, **maintainability**, and **performance**.
- Common refactoring techniques include:
  - Renaming variables for clarity
  - Breaking large functions into smaller ones
  - Removing duplicate code
- Helps reduce technical debt and improve long-term project health.

Cohesion

Cohesion refers to how closely related the responsibilities of a module or class are.

Types of Cohesion (from low to high):

- **Coincidental**: Random tasks grouped together.
- **Logical**: Tasks of similar nature but not related.
- **Functional**: All elements contribute to a single well-defined task.

High Cohesion:

- Modules perform a single, focused task.
- Easier to understand, test, and maintain.

Coupling

Coupling refers to the degree of interdependence between modules.

Types of Coupling (from high to low):

- **Content Coupling**: One module directly modifies another.
- **Control Coupling**: One module controls the flow of another.

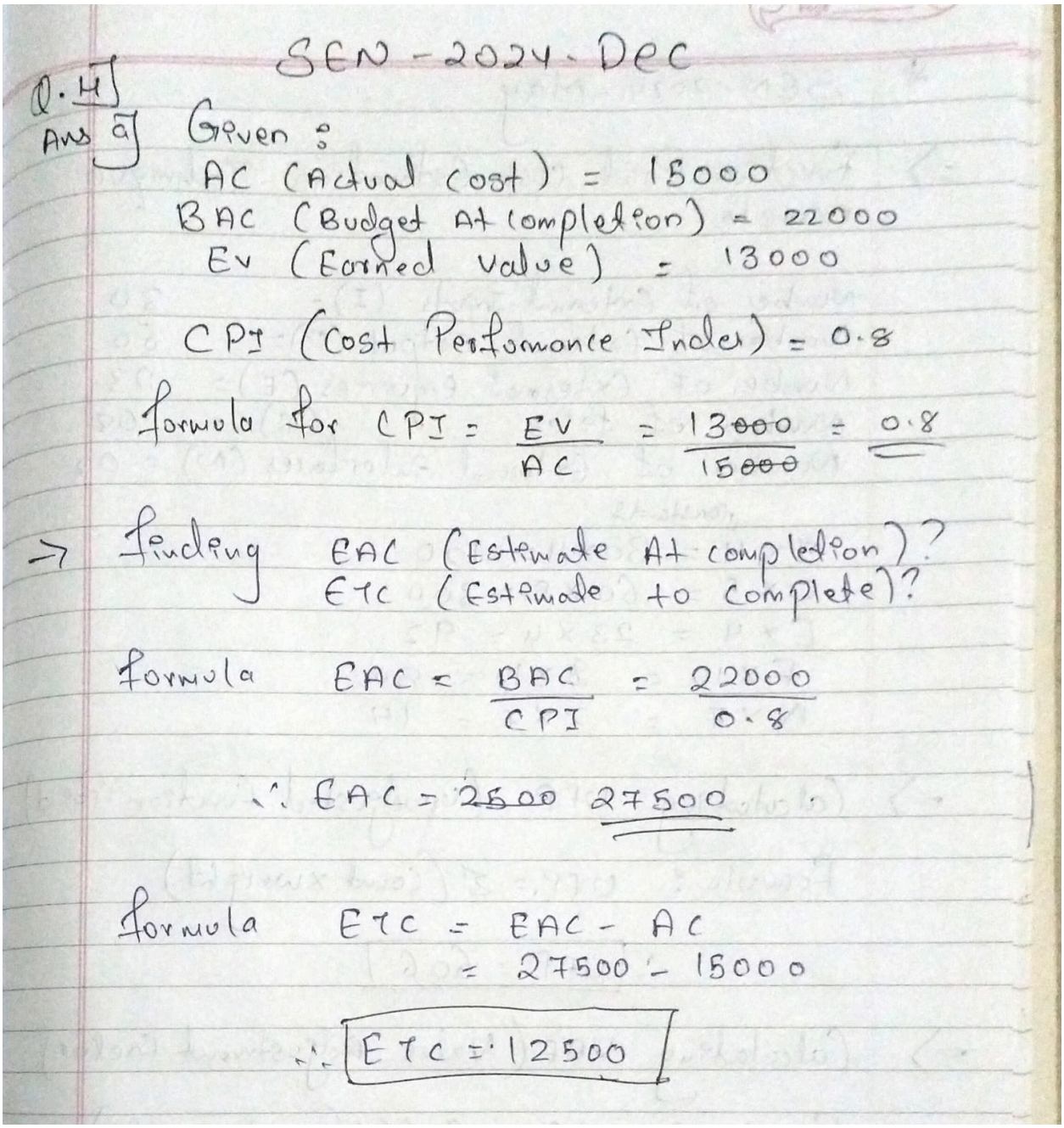
- **Data Coupling**: Modules share data through parameters.

Low Coupling:

- Modules are independent and interact through well-defined interfaces.
- Enhances flexibility and reusability.

Q4. [10 Marks] - Answers

a. Explain Earned Value Analysis? Find project EAC, ETC, where AC is 15000, BAC is 22000, EV is 13000, CPI is 0.8?



Q5. [10 Marks] - Answers

a. Explain about Project scheduling and anyone tracking Technique with example?

What Is Project Scheduling?

Project scheduling involves creating a timeline for project activities, assigning resources, and setting milestones. It ensures that tasks are completed in the right order and within the allocated time and budget.

Key Elements of Project Scheduling:

- **Task Identification**: Break down the project into manageable tasks.
- **Sequencing**: Determine task dependencies (what comes before or after).
- **Duration Estimation**: Estimate time required for each task.
- **Resource Allocation**: Assign people, tools, and budget.
- **Milestones**: Define key checkpoints or deliverables.

Tracking Technique: Gantt Chart

A **Gantt Chart** is a visual tool used to track project progress over time. It displays tasks along a timeline, showing start and end dates, dependencies, and current status.

Features of Gantt Chart:

- Horizontal bars represent tasks.
- Length of each bar indicates task duration.
- Dependencies are shown with arrows.
- Progress is tracked by shading completed portions.

Example:

Imagine an **Online Shopping Website** project with the following tasks:

Task	Duration	Start Date	End Date
Requirement Analysis	5 days	Nov 1	Nov 5

Task	Duration	Start Date	End Date
UI Design	7 days	Nov 6	Nov 12
Backend Development	10 days	Nov 13	Nov 22
Testing	5 days	Nov 23	Nov 27
Deployment	3 days	Nov 28	Nov 30

In a Gantt chart:

- Each task is shown as a bar across the timeline.
- Dependencies (e.g., Testing starts after Development) are linked.
- Progress updates help track delays or early completions.

Q6. [10 Marks] - Answers

a. Explain the principles of software testing? Discuss in detail the difference between black box and white box testing technique?

Principles of Software Testing

Software testing is guided by several key principles that ensure effective defect detection and quality assurance:

1. **Testing Shows Presence of Defects**

- Testing can reveal defects but cannot prove that the software is defect-free.

2. **Exhaustive Testing Is Impossible**

- It's impractical to test all possible inputs and paths; focus on risk-based and priority testing.

3. **Early Testing Saves Time and Cost**

- Begin testing during the requirement and design phases to catch issues early.

4. **Pesticide Paradox**

- Repeating the same tests will not find new bugs; test cases must be regularly reviewed and updated.

5. **Testing Is Context Dependent**

- Testing approach varies based on application type (e.g., web app vs. embedded system).

Black Box vs. White Box Testing – Comparison Table

Feature	**Black Box Testing**	**White Box Testing**
Definition	Tests software functionality without knowing internal code	Tests internal logic, structure, and code paths
Focus Area	Input/output behavior	Control flow, loops, conditions, branches
Knowledge Required	No programming knowledge needed	Requires programming and code-level understanding
Test Basis	Requirements and specifications	Source code and design
Techniques Used	Equivalence partitioning, boundary value analysis	Statement coverage, branch coverage, path coverage
Performed By	Testers or end users	Developers or technical testers
Advantages	Simulates real user scenarios	Ensures code correctness and optimization
Limitations	May miss internal errors	May not detect missing functionalities