Part – B

**Justification: “Design is not coding and coding is not design”**

The statement **"Design is not coding and coding is not design"** highlights a **fundamental distinction** between two key phases of software development: **software design** and **implementation (coding)**. they serve **different purposes**, involve **different levels of abstraction**, and require **different skill sets**.

**🔷 1. Software Design: A Planning Phase**

* **Design** refers to the **process of defining the architecture, components, interfaces, and data** for a system to satisfy specified requirements.
* It is **independent of programming languages**.
* focuses on **structure**, **relationships**, and **responsibilities** of components.
* **Output**: High-level design diagrams, component specifications, flowcharts, UML models.

✅ **Example**:  
Suppose you're developing an online shopping application. During design:

* You decide to divide the system into modules like *User Management*, *Product Catalog*, *Shopping Cart*, and *Payment Gateway*.
* You define how these components will interact and what data each will handle.
* This is done using **UML diagrams, ER diagrams, or architecture diagrams**, not code.

**🔷 2. Coding: A Translation Phase**

* **Coding** is the process of **converting the design into executable code** using a programming language.
* It involves **syntax, language constructs**, and optimization for **performance or maintainability**.
* **Output**: Functional software written in code (e.g., Java, Python, C++).

✅ **Example**:  
Continuing the shopping app example, during coding:

* You implement the *addToCart()* function in Java.
* You write SQL queries for product retrieval.
* You use HTML/CSS to create the product page layout.

What are the different types of architectural styles exist for software and explain any one software architecture. 08

Ans: Common architectural styles include:

* **Layered Architecture**
* **Client–Server Architecture**
* **Repository Architecture**
* **Model–View–Controller (MVC) Architecture**
* **Pipe and Filter Architecture**
* **Event-Driven Architecture**
* **Service-Oriented Architecture (SOA)**
* **Microservices Architecture**

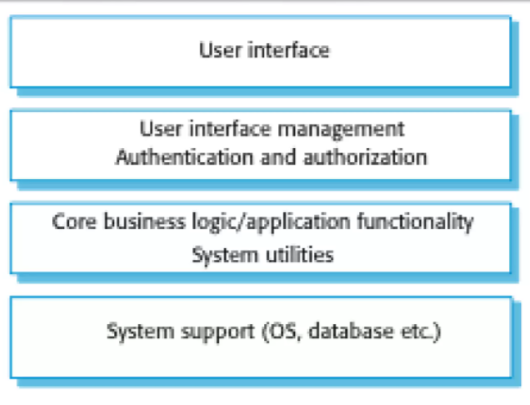
**✅ Explanation of One Style: Layered Architecture**

**🔷 Definition:**

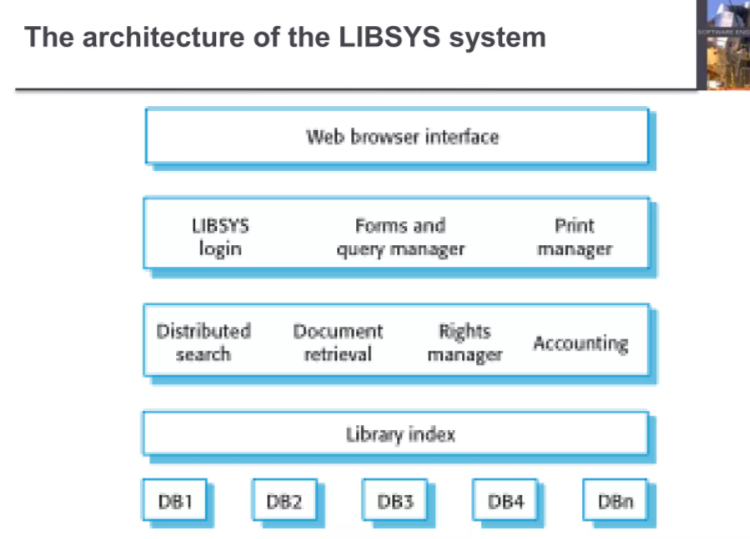
In **Layered Architecture**, the system is organized into a **hierarchical set of layers**, where **each layer provides services to the layer above** and **relies on services from the layer below**.

**🔷 Common Layers:**

1. **Presentation Layer** – Handles UI and user interaction.
2. **Application/Business Logic Layer** – Contains core functionality and logic.
3. **Data Layer** – Manages access to databases or file storage.



Example: Library system Acrhitecture



**✅ Architecture of the LIBSYS System (Simplified):**

1. **User Interface Layer**
   * Users access the system through a **web browser interface**.
2. **Application Management Layer**
   * Handles user login, forms for queries, and printing through components like:
     + **LIBSYS login**
     + **Forms and query manager**
     + **Print manager**
3. **Service Processing Layer**
   * Manages core services such as:
     + **Distributed search**
     + **Document retrieval**
     + **Rights management**
     + **Accounting**
4. **Indexing Layer**
   * Maintains the **Library index**, which helps in efficient searching and organizing of records.
5. **Data Storage Layer**
   * Connects to multiple **databases (DB1 to DBn)** that store actual library data (books, users, records).

Discuss in detail the importance of design process in software development. 10 marks

**1. Bridges Requirements and Implementation**

* The design phase acts as a **link between requirements analysis and coding**.
* Example: In the packing robot system (Figure 6.1), the design defines how the **Vision System** communicates with the **Arm Controller**.

**2. Ensures System Quality (Non-Functional Requirements)**

* A well-thought-out design directly impacts:
  + **Performance** (e.g., optimized component interactions).
  + **Scalability** (e.g., modular components for future expansion).
  + **Reliability** (e.g., error-handling mechanisms).
  + **Maintainability** (e.g., clean separation of concerns).

**3. Facilitates Stakeholder Communication**

* Architectural models (e.g., block diagrams) help **non-technical stakeholders** (clients, managers) visualize the system.
* Example: A high-level design of a **banking system** helps managers allocate resources efficiently.

**4. Supports Reusability and Modularity**

* Good design promotes **component-based development**, allowing reuse across projects.
* Example: A **login authentication module** can be reused in multiple applications.

**5. Reduces Development Costs & Risks**

* Identifying flaws early (e.g., bottlenecks, security gaps) prevents expensive fixes later.
* Example: A poorly designed **database schema** can slow down an entire application.

**6. Enables Efficient Team Collaboration**

* Clear design documentation ensures developers, testers, and DevOps teams work cohesively.
* Example: **Microservices architecture** allows teams to work independently on different services.

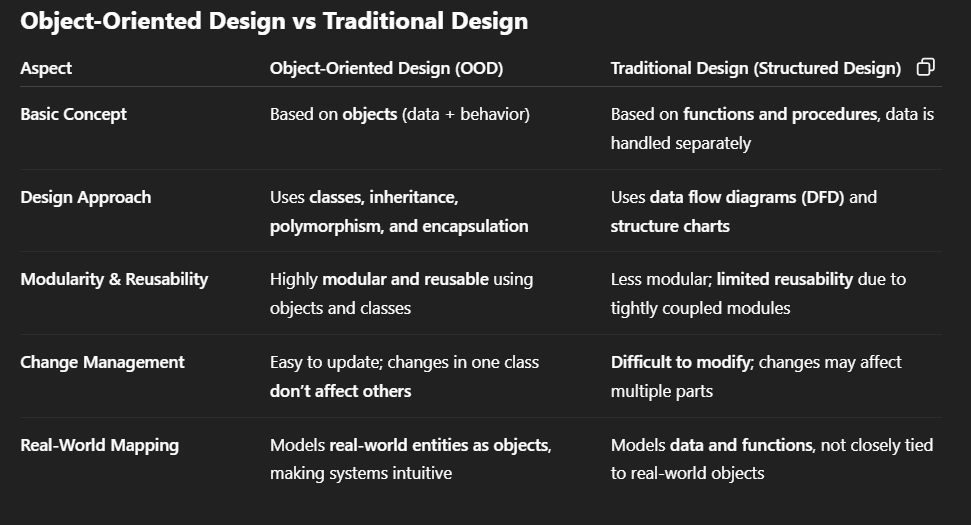
**7. Enhances Testability**

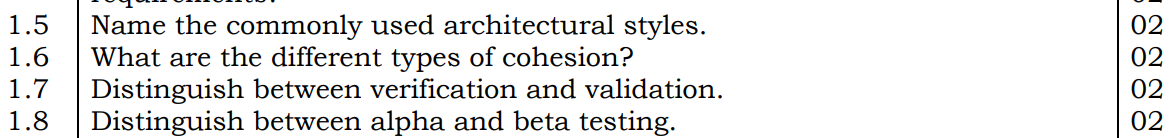
* A structured design makes it easier to **write unit tests, integration tests, and system tests**.
* Example: A **well-defined API contract** simplifies automated testing.

**9. Ensures Security & Compliance**

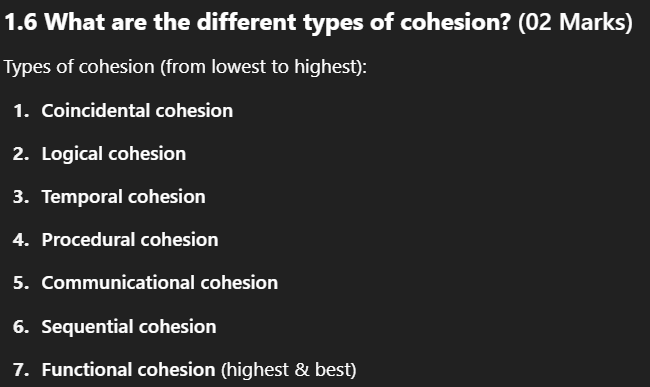
* Security considerations (e.g., encryption, access control) must be embedded in the design phase.
* Example: A **healthcare app** must comply with **HIPAA** regulations in its design.

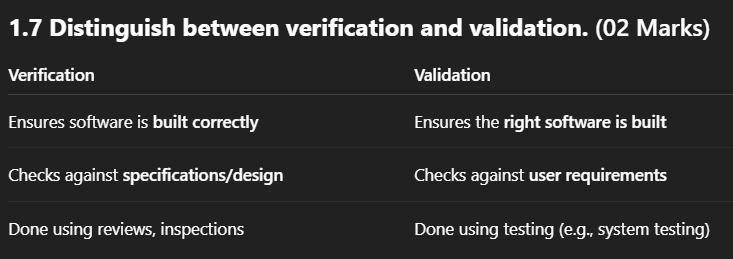
b Write short notes on “object oriented design” as compared to traditional design. 08

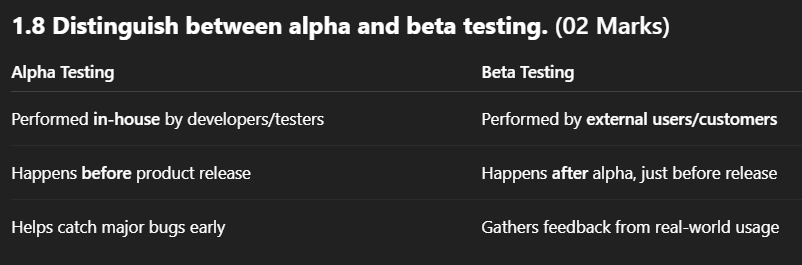


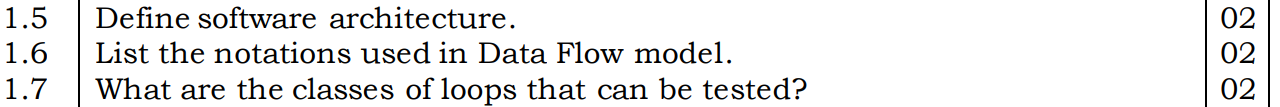












Ans:

**1.5 Define Software Architecture.**  
**(Marks: 02)**  
Software architecture is the **structured framework** used to conceptualize software elements, relationships, and properties. It defines the **high-level structure** of a software system, including its **components**, **modules**, **interfaces**, and **data flow**, and how they interact to fulfill requirements.

**1.6 List the notations used in Data Flow Model.**  
**(Marks: 02)**  
Common notations used in the Data Flow Model include:

* **Processes** – represented by **circles** or **bubbles**
* **Data Flows** – represented by **arrows**
* **Data Stores** – represented by **open-ended rectangles** or **parallel lines**
* **External Entities** – represented by **rectangles**

**1.7 What are the classes of loops that can be tested?**  
**(Marks: 02)**  
The classes of loops that can be tested include:

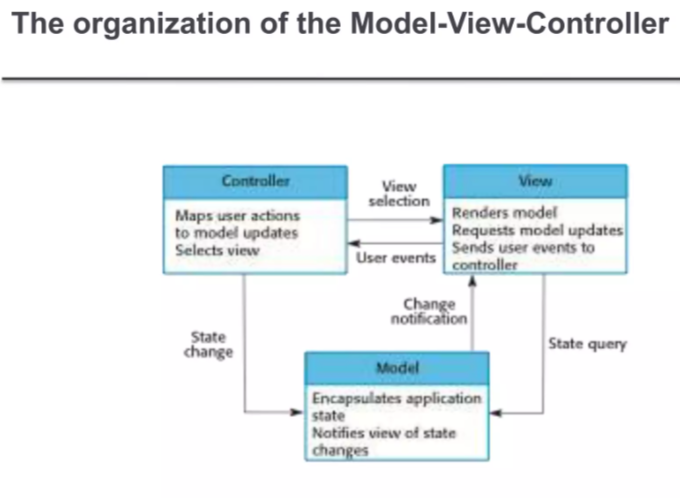
* **Simple Loops** – loops with a single entry and exit point (e.g., for, while)
* **Nested Loops** – loops inside other loops
* **Concatenated Loops** – multiple loops executed one after another
* **Unstructured Loops** – loops with multiple entry or exit points (usually due to goto or complex conditions)

6 Explain in detail about any four architectural styles for system design. 16 marks

Ans:

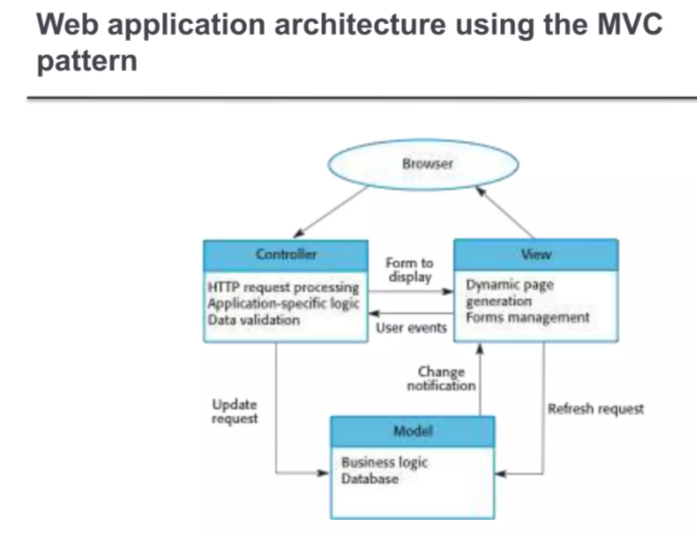
Layered style : already in the previous

MVC (Model View Controlller):



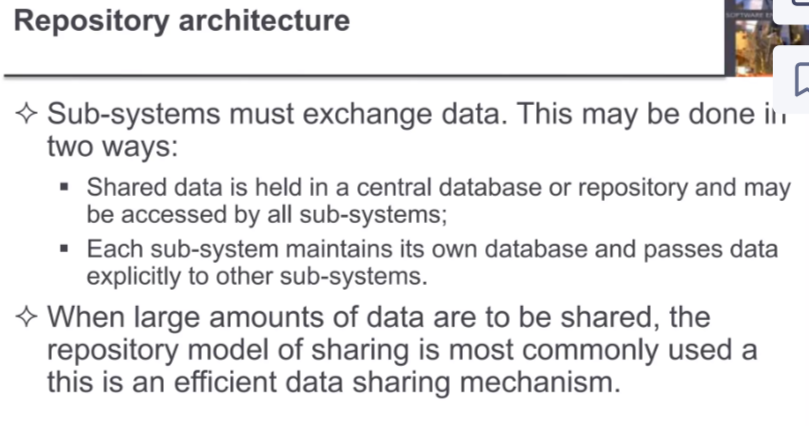
* **Controller** maps user actions to model updates and selects the view.
* **View** renders the model, sends user events to the controller, and may request updates.
* **Model** manages application state, notifies views of changes, and handles state queries.
* **User events** trigger controller actions, leading to model updates.
* **State changes** in the model notify the view to refresh.
* **MVC ensures separation of concerns** for modular and maintainable design.
* **State Query & Change Notification**: The Model handles state queries from the View/Controller and sends change notifications to the View for updates.

Example:

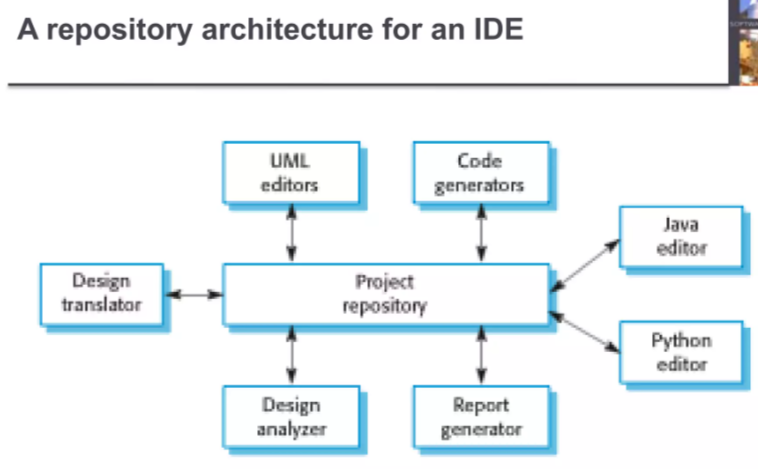


* **Model-View-Controller (MVC)** separates application logic into three interconnected components.
* **Controller** handles HTTP requests, input validation, and invokes the model.
* **Model** contains business logic and interacts with the database.
* **View** manages dynamic page generation and form handling for user interfaces.
* **User events** from the browser are passed to the controller and then to the model.
* **Model updates** trigger changes in the view, keeping the UI in sync with the data.

Repository Architecture:

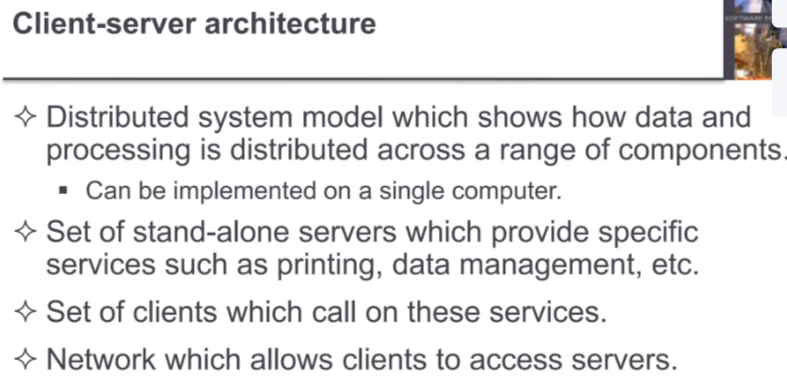


Example:

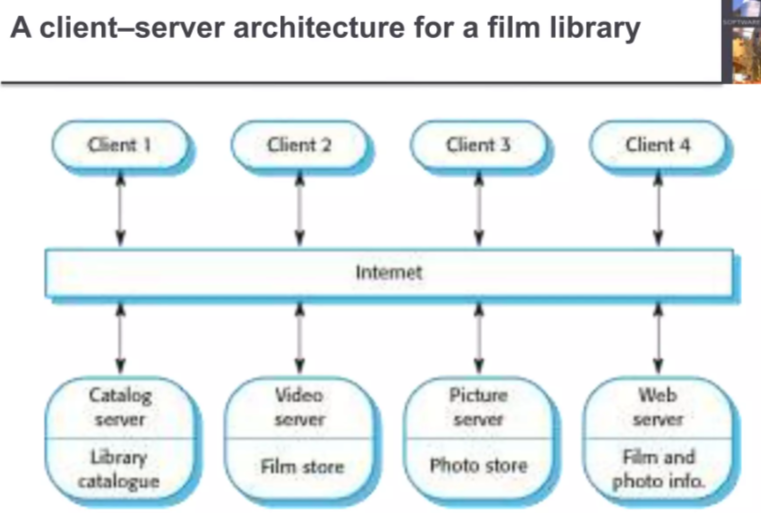


1. **Central Repository** acts as the core data store for all IDE components and project files.
2. **Code Editors** (Java, Python) allow users to write and modify source code directly.
3. **Code Generators** automate the creation of code snippets or boilerplate templates.
4. **Design Translator** converts design models (e.g., UML) into executable code.
5. **Design Analyzer** checks code structure, dependencies, or design patterns for errors.
6. **Report Generator** produces documentation, metrics, or analysis summaries.
7. **Project Repository** manages version control, file history, and collaborative workflows.

Client – Server Architecture.



Example



1. **Multiple Clients** (Client 1-4) access the film library system remotely via the internet.
2. **Catalog Server** manages the central library catalogue for organizing film and photo data.
3. **Video Server** stores and streams films from the film store to clients.
4. **Picture Server** handles photo storage and retrieval from the photo store.
5. **Web Server** provides film/photo information and interfaces for client access.
6. **Centralized System** ensures efficient data management and resource sharing across all clients.

What are the characteristics of a real time system? Explain why real time systems usually have to be implemented using concurrent process.08 marks

Ans:

**Characteristics of Real-Time Systems (5 points)**

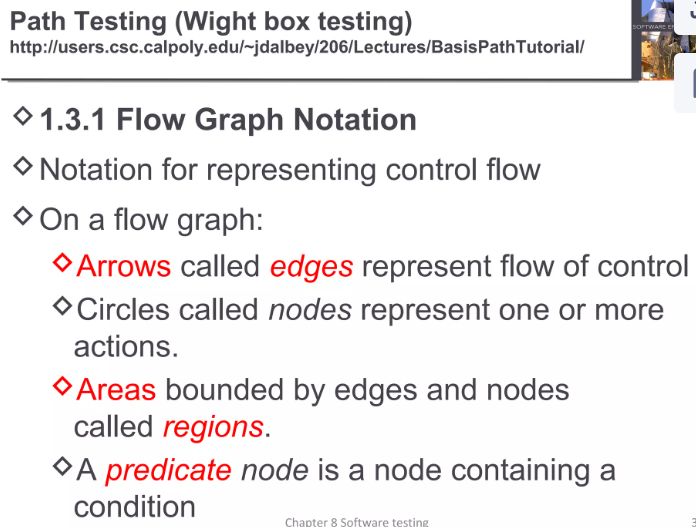
1. **Timing Constraints** – Must respond within strict deadlines (hard real-time) or acceptably fast (soft real-time).
2. **Deterministic Behavior** – Predictable response times to ensure reliability in critical operations.
3. **Event-Driven** – Reacts to external events (e.g., sensors, user inputs) immediately.
4. **High Reliability** – Fault tolerance and minimal downtime are essential (e.g., medical systems, aviation).
5. **Concurrency** – Handles multiple tasks simultaneously to meet timing requirements.

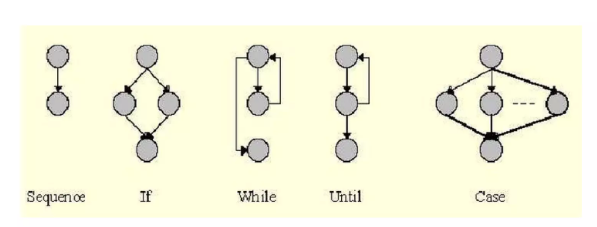
**Why Concurrent Processes Are Used (5 points)**

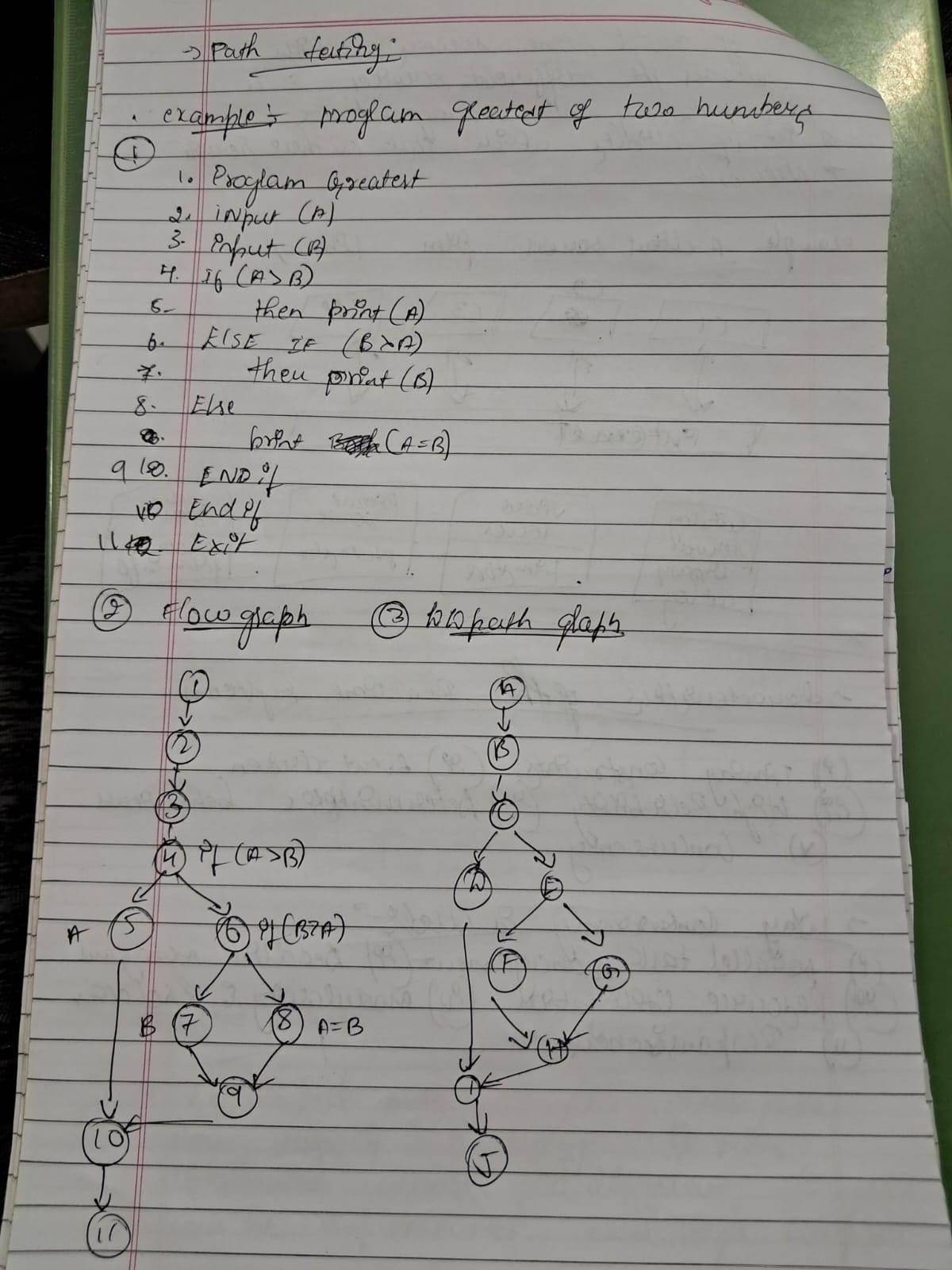
1. **Parallel Task Handling** – Multiple processes run simultaneously to manage different real-time tasks efficiently.
2. **Deadline Management** – Concurrency ensures critical tasks meet deadlines while non-critical ones run in parallel.
3. **Resource Utilization** – Optimizes CPU usage by allowing overlapping I/O and computation.
4. **Modularity & Isolation** – Prevents failures in one task from crashing the entire system.
5. **Responsiveness** – Enables immediate reaction to high-priority events without blocking lower-priority processes.

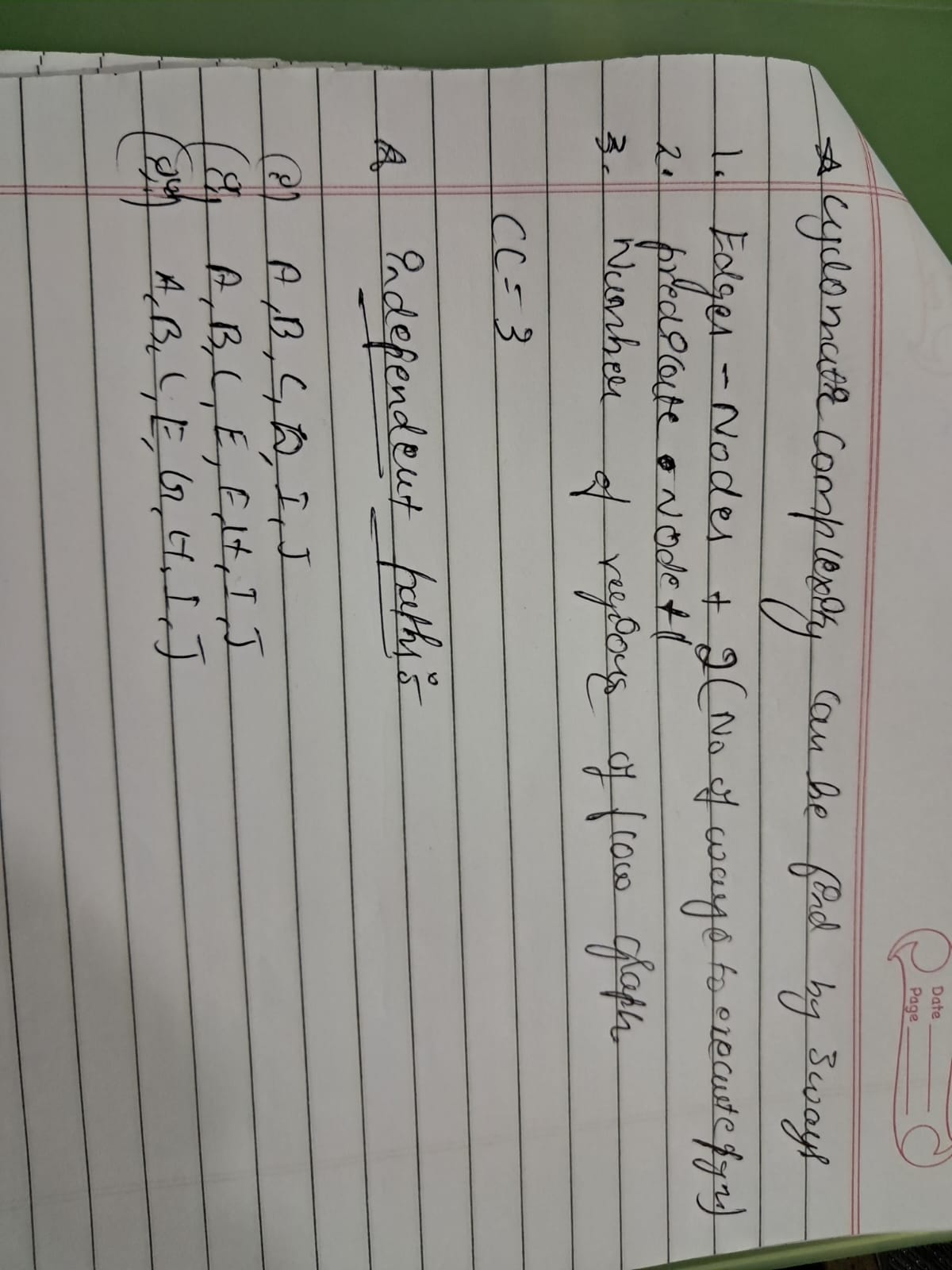
Explain in detail the steps involved in Basis Path Testing with appropriate flow graphs wherever required. 16 marks

Ans:









1.6 Define sub system model and sequence model. 02

1.7 With a neat diagram, write the debugging process. 02

1.8 Define component testing and integration testing. 02

Ans:

**1.6 Define Subsystem Model and Sequence Model**

* **Subsystem Model:** It shows how a system is organized into logically related subsystems or modules.
* **Sequence Model:** It represents the interactions between objects or components in a time-sequenced order during a specific scenario.

**1.7 Debugging Process (with diagram)**

* **Debugging** is the process of identifying, isolating, and fixing errors in software.

**Steps (with neat diagram):**

1. **Error Detection → Error Location → Error Analysis → Error Correction → Retesting**

**1.8 Define Component Testing and Integration Testing**

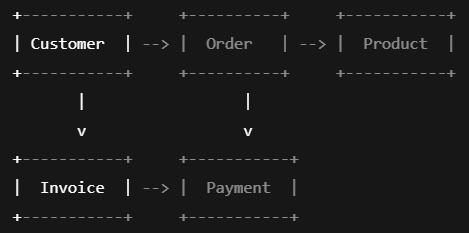
* **Component Testing:** It verifies individual components or modules in isolation for correctness.
* **Integration Testing:** It tests the interaction between integrated components to ensure they work together correctly.

b With a neat block diagram, explain the object oriented decomposition for invoice processing sub system. 06 marks .

Ans:

**✅ Object-Oriented Decomposition for Invoice Processing – Single-line Points:**

1. **Object-Oriented Decomposition** breaks down the system into interacting objects, each with responsibilities.
2. In **invoice processing**, common objects include: Customer, Invoice, Order, Product, and Payment.
3. **Customer object** stores customer info and links to order and invoice.
4. **Order object** maintains list of items/products to be billed.
5. **Invoice object** calculates total cost, taxes, and generates bill.
6. **Product object** includes details like product name, ID, price, and stock.
7. **Payment object** handles billing status, payment type, and receipt generation.



Illustrate with the aid of an appropriate example how to design a real time monitoring and control system pending

c Bring out the advantages of shared repository. 04

1. Enables centralized access to the project for all team members.
2. Maintains complete version history of code changes.
3. Supports efficient collaboration and teamwork.
4. Helps in detecting and resolving code conflicts.
5. Provides secure backup and controlled access to the code.

7 a With an example, explain nay two object oriented models. 08 marks pending

Explain the importance of verification and validation during the development process and illustrate how this is achieved in the V-model of software development. (8 marks)

Ans:

**✅ Importance of Verification and Validation (V&V)**

1. **Verification** ensures the product is built **correctly** as per design.
2. **Validation** ensures the final product meets **user requirements**.
3. V&V help detect **errors early**, reducing cost and time.
4. They improve **software quality, reliability, and user satisfaction**.



**✅ V-Model Achieves V&V Through:**

1. Each **development phase** has a corresponding **testing phase**.
2. **Unit tests** verify code modules after module design.
3. **Integration tests** validate system structure after architecture design.
4. **System and acceptance tests** validate complete system functionality.

b Explain two distinct phases of system testing. 06

Ans:

**1. Functional Testing Phase**

📌 *Checks what the system does — does it behave as expected?*

**🔹 Key Points:**

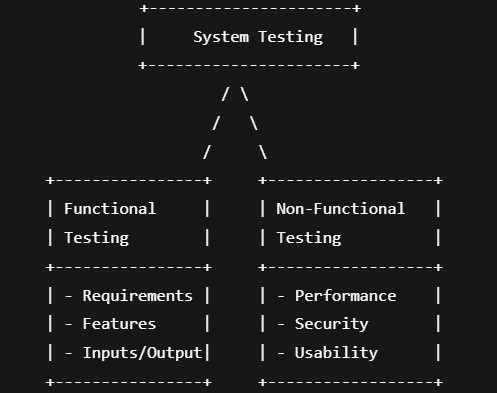
1. Tests the actual **functions** of the system against requirements.
2. Uses **black-box testing** — testers don’t need to know internal code.
3. Validates inputs, outputs, and user interactions.
4. Ensures the system performs **all intended operations correctly**.
5. Example: Login, file upload, payment processing.

**2. Non-Functional Testing Phase**

📌 *Checks how the system performs — speed, security, etc.*

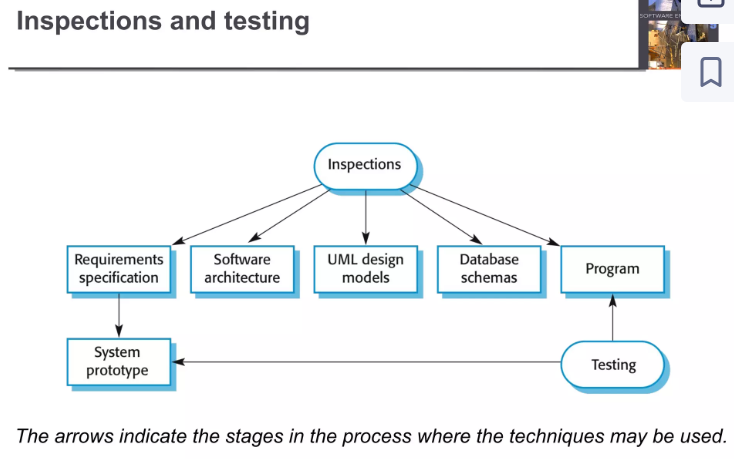
**🔹 Key Points:**

1. Evaluates the **quality attributes** of the system.
2. Includes **performance, security, usability, and compatibility** tests.
3. Ensures the system performs well under expected and stress loads.
4. Helps identify issues related to **response time, scalability**, etc.
5. Example: Load testing a website with 1000 users.



9 a Explain with block diagram of inspection process and also give major advantages over testing

Ans:



1. **Inspections** involve formal reviews of documents and code to detect defects early.
2. **Requirements specification** is inspected to ensure clarity, completeness, and correctness.
3. **Software architecture** is reviewed to verify structural integrity and alignment with requirements.
4. **UML design models** are checked for consistency and accuracy in system representation.
5. **Database schemas** are inspected for proper design, normalization, and efficiency.
6. **Program code** is reviewed to identify syntax errors, logic flaws, and adherence to standards.
7. **System prototype** is tested to validate functionality and user requirements.
8. **Testing** is conducted at various stages to ensure the system works as intended.
9. **Arrows** indicate the stages where inspection and testing techniques are applied.
10. **Early defect detection** through inspections reduces later development costs.

**Major Advantages of Inspections Over Testing**

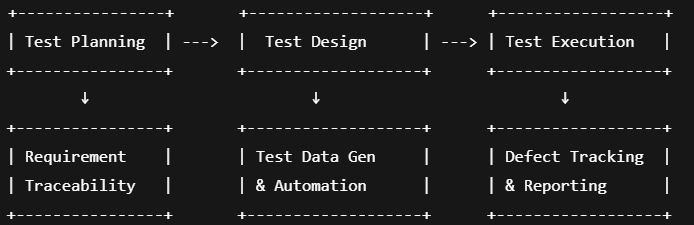
1. **Early Defect Detection** – Finds issues in requirements/design before coding begins.
2. **Cost-Effective** – Fixing defects early is cheaper than post-development fixes.
3. **Improves Documentation** – Ensures clarity and correctness of specs/designs.
4. **Identifies Logical & Design Flaws** – Catches errors that testing may miss (e.g., incorrect logic, scalability issues).
5. **Reduces Testing Effort** – Fewer defects reach the testing phase, saving time.
6. **Enhances Software Quality** – Proactive defect prevention leads to more reliable systems.

b Discuss about the software testing work bench with a diagram. 08 marks

Ans:

**✅ Software Testing Workbench – Simple Points**

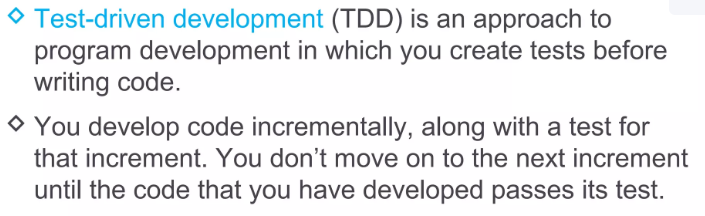
1. A **Software Testing Workbench** is a structured environment for planning, designing, and executing tests.
2. It supports the **complete testing process** — from test planning to reporting.
3. Ensures **consistency, accuracy, and repeatability** in software testing.
4. Provides **tools and guidelines** for each testing phase.
5. Enhances **automation, documentation, and defect tracking**.
6. Improves collaboration between **developers and testers**.
7. Helps in **early error detection**, reducing cost and time.
8. Often includes **test case design tools, test data generators, and execution frameworks**.
9. Supports various types of testing: **unit, integration, system, and acceptance**.
10. Ensures better **traceability** between requirements and test cases.

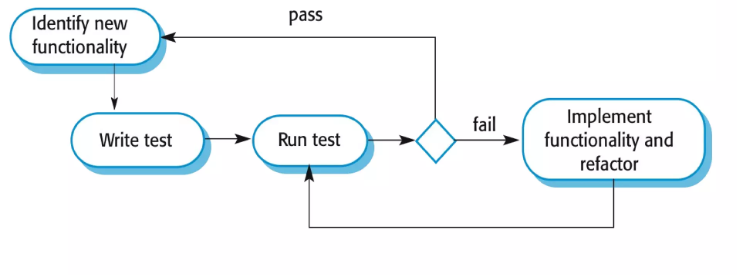


Discuss two types of Model driven architecture with an example pending

b Explain the concept of test-driven development. How does it contribute to software quality?

Ans:





**Concept of TDD (Using Given Diagram):**

1. **Identify New Functionality** – Define a small, testable feature requirement.
2. **Write Test First** – Create a failing test case *before* writing implementation code. *(✓ in diagram)*
3. **Run Test & Fail** – Verify the test fails (confirms the test is valid). *(✓ in diagram)*
4. **Implement & Refactor** – Write minimal code to pass the test, then optimize without breaking functionality. *(✓ in diagram)*

**Contribution to Software Quality:**

* **Early Bug Detection** – Tests catch defects immediately during development.
* **Improved Design** – Forces modular, loosely coupled code for testability.
* **Documentation** – Tests serve as executable specs for expected behavior.
* **Regression Safety** – Automated tests prevent new changes from breaking existing features.
* **Confidence in Refactoring** – Ensures code improvements don’t introduce errors.

6 a How do architectural views assist in managing the complexity of software design? Provide examples for architectural views. 08

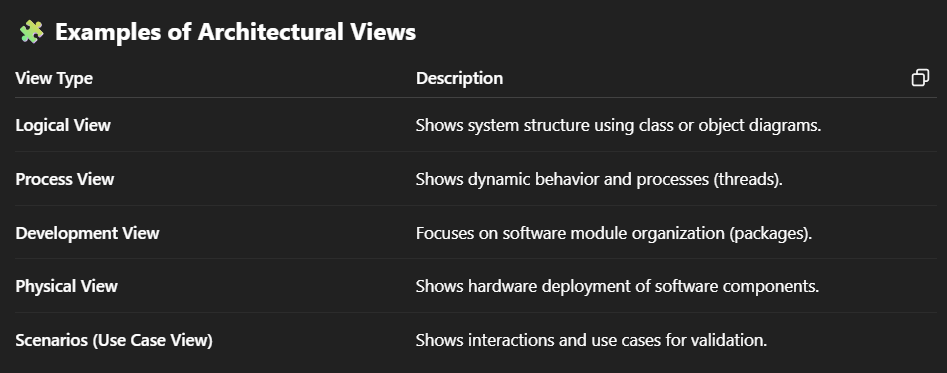
Ans:

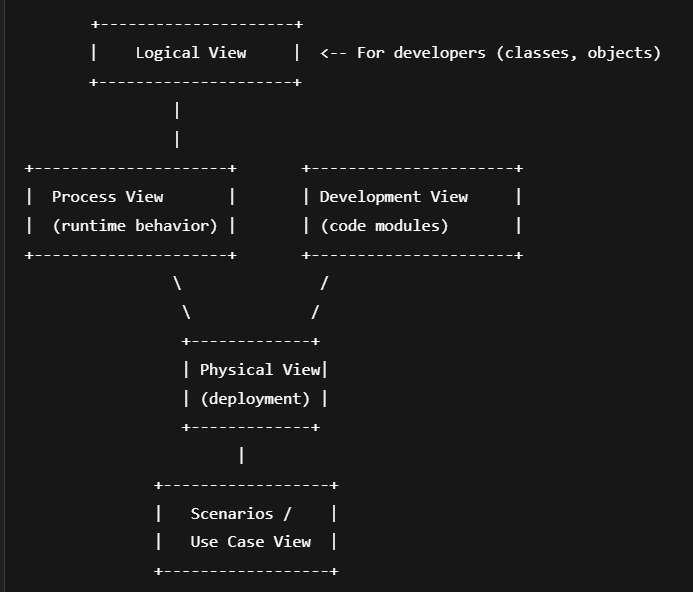
**✅ How Architectural Views Help in Managing Software Complexity**

**Architectural views** break down complex software systems into multiple perspectives, helping different stakeholders understand and work on the system efficiently.

**🔹 Simple Single-Line Points (8 Points):**

1. Architectural views **divide complex systems** into manageable parts.
2. They help **different stakeholders** (developers, users, testers) focus on relevant aspects.
3. Reduce confusion by providing **clear separation of concerns**.
4. Enable **parallel development** by different teams.
5. Improve **communication and documentation** among team members.
6. Help identify and fix **design issues early**.
7. Allow better **traceability from requirements to design**.
8. Support **reusability and maintenance** by modular design.





b Discuss the challenges of open source software development. How do these challenges impact the quality and adoption of open- source projects? 08

Ans:

**✅ Challenges of Open Source Software Development *(Single-line points)***

1. **Lack of formal documentation** makes understanding and contribution difficult.
2. **Inconsistent code quality** due to varied contributor skill levels.
3. **Limited funding** leads to slower development and fewer resources.
4. **Security vulnerabilities** may go unnoticed without strict review.
5. **Fragmented community** can cause project direction conflicts.
6. **Dependency on volunteers** leads to unpredictable maintenance.
7. **Lack of support services** may discourage enterprise adoption.
8. **Difficulty in governance** may result in poor decision-making.

**✅ Impact on Quality and Adoption**

1. Poor documentation and quality may **reduce trust** in the software.
2. Security risks make organizations **hesitate to adopt** it.
3. Inactive or poorly maintained projects can lead to **abandonment**.
4. Lack of support affects **business continuity and integration**.
5. Fragmented or unclear goals may **limit contributions and users**.

1.2 What are Architectural design, Database design, Interface design and Component selection and design? 02

1.3 List and briefly explain any 2 types of UML diagrams. 02

1.4 State any four Extreme Programming techniques. 02

Ans:

**✅ 1.2 Software Design Types**

**🔹 1. Architectural Design (2 marks)**

1. Defines the **overall structure** of the software system.
2. Focuses on how components interact, including **architecture styles** like layered or client-server.

**🔹 2. Database Design (2 marks)**

1. Organizes **data storage structures**, like tables and relationships.
2. Ensures **data consistency**, integrity, and optimized access.

**🔹 3. Interface Design (2 marks)**

1. Specifies how users and systems **interact with software**.
2. Includes **UI design** and **APIs** between modules.

**🔹 4. Component Selection and Design (2 marks)**

1. Involves choosing **reusable software components** or libraries.
2. Designs each component’s **functions and interactions**.

**✅ 1.3 Two Types of UML Diagrams (2 marks)**

**🔹 1. Use Case Diagram (1 mark)**

1. Shows **user interactions** with the system using actors and use cases.
2. Useful for **requirement gathering**.

**🔹 2. Class Diagram (1 mark)**

1. Represents **classes, attributes, and relationships**.
2. Used in **object-oriented design**.

**✅ 1.4 Four Extreme Programming (XP) Techniques (2 marks)**

1. **Pair Programming** – Two developers write code together at one workstation.
2. **Test-Driven Development (TDD)** – Write tests before writing code.
3. **Continuous Integration** – Frequently merge code changes into a shared repository.
4. **Refactoring** – Continuously improve existing code without changing its behavior.