Name- Akhil Kumar Univ. Roll No - 2021066 ①
Section- E
Roll NO - 8

Q1- What are the top-down and botton up approaches in software designs Compare their advantages and disadvantages.

Ans- In software design, Top-Down and Bottom up are two fundamental strategies used to plan and build systems.

- Top-Down Approach

In this design begins at the highest level of the system. One first visualize the complete system as a whole, then gradually break it down into smaller and more specific components or modules. It's like outlining a big story and then working out each chapter one by one.

Advantages
- Encourages better understanding of system goals
- Maintains clarity in architecture and flow.

Disadvantages
- Lower level details might get ignored.
- Not ideal when reusable components already exist.

- Bottom-Up Approach

In this one design and develop the basic building blocks first. small modules or components. These smaller units are then combined to form larger subsystems, eventually assembling the full application.

Eg: You first create independent modules like a payment gateway, product display card, and login form, then integrate them to form a complete e-commerce system.

Advantages
- Promotes reusability of components
- Easier to test small parts early.

Disadvantages
- can lose sight of the overall system goal.
- integration of components may become challenging.

Q2- Explain the concept of Object - Oriented Design and its role in modern software development. illustrate with example

Ans- Object - Oriented Design is a software design approach that revolves around the idea of modeling real-world entities as objects. These objects bundle data and behavior into a single unit. This makes software more modular, easier to manage, and closer to how human naturally perceive and organize the world.

In OOD we use the principle of classes, inheritance, encapsulation, polymorphism and abstraction to build systems that are flexible and maintainable

Importance of OOD in modern Software development

- Reusability: Once a class is written, it can be reused in multiple parts of a program or even in other projects.
- Maintainability: changes can be made to one part of the code without affecting the whole system.
- Scalability: OOD makes it easier to scale software as more features or requirements are added
- Modularity: The code is divided into small, manageable chunks that are easier to debug and test.

Example: E commerce System

Let's say we are building an online store

Step1. Identify objects

- Product
- User
- cart

Step2: Define classes: each representing a real world object

```
class Product {        class User {

    };                     };
```

Benefits in this case

- if you want to change how product is displayed, you only update displayDetails() method.
- you can reuse the Product class in other parts of the system, like inventory or recommendations.

Q3- Define Software design patterns. Describe the types of design patterns with relevant examples.

Ans- Software design patterns are tried and tested solutions to common problems that developers face while designing software systems. Think of them as templates or blueprints that guide one in building parts of ~~ones~~ applications efficiently and in a structured way- without reinventing the wheel every time.

These patterns do not give one complete code but rather show how different components should interact to solve a ~~problem~~ particular type of problem.

Type of Design Patterns

1- Creational Pattern: These patterns deal with object creation- helping you control how objects are created in a system especially when creating many objects or ensuring they follow a specific structure.

Eg: singleton Patter

2- Structural Patterns: These focus on how classes and objects are composed to form larger structure, making it easier to ensure components fit together smoothly

Eg: Adapter Pattern

3- Behavioral Pattern

These patterns define how objects communicate with each other and manage responsibilities or workflows.

Eg. Observer Pattern

Q4. Discuss the impact of d/f software architectures on system performance and maintainability. Provide real-world example.

Ans- Impact of Software architectures on System performance and maintainability

Software architecture refers to the high level structure of a software system- how components are organized and how they interact. The chosen architecture plays a major role in how the system performs under load and how easily it can be maintained or updated over time.

1. Monolithic Architecture

In this architecture, the entire application is built as a single, unified unit.

Impact on Performance:

- Efficient when handling simple or small-scale app.
- Fast internal communication since everything runs in one process.

Impact on maintainability

- Becomes harder to modify as it grows
- A small change might require re-deploying the entire system.
- Codebase becomes tightly coupled and difficult to manage.

Real world Example

Early versions of WordPress and Amazon were built as monoliths before scaling issues led to architectural shifts.

2. Microservices Architecture:

Here the app is broken into small, independent services, each responsible for a specific task and they communicate via APIs

Impact on performance:

- can be scaled independently.
- Handles high traffic better when distributed properly

Impact on Maintainability

- Services are loosely coupled, making it easier to update or fix parts of the system without affecting the rest.
- D/f teams can work on d/f services in parallel.

• Real world Example

• Netflix transitioned to microservices to handle massive global demand and frequent updates

3- Layered (N-tier) Architecture •

This design seperates concerns into layers - like presentation, buisness logic, and data access.

Impact on Maintainability

· Well organized and modular, making code easier to update & test.

• Each layer can be developed and maintained independently.

Impact on Performance

• Might introduce latency due to layer-by-layer communication

• Can be slower compared to lightweight architectures for performance-critical apps.

Real world Example

• commonly used in banking software, educational portals.

4- Event driven Architecture

In this style, components communicate through events - each component reacts to specific events triggered in the system

Impact on performance

• Highly responsive and scalable, especially in real-time systems.

• Reduce unnecessary processing by reacting only when needed.

Impact on Maintainability

• Promotes loose coupling.

· Easier to modify and replace individual even-handlers.

Real world Example

• Uber uses event-driven architecture for live tracking and updates.

Q5- Explain the different types of software testing techniques How does software testing contribute to software quality assurance?

Ans- Types of Software Testing techniques

1- Unit testing: Tests individual units or components of a software.

Goal: Ensure each part behaves correctly in isolation.

Eg: Testing a function that calculates the total price after discount.

2- Integration Testing: Tests how different modules or services interact.

Goal: Detect issues when components are combined.

Eg: Checking whether the login module correctly connects with the database.

3- System testing: Tests the complete software as a whole.

Goal: Validate the end-to-end flow of the app.

Eg: testing an e-commerce site from searching a product to placing an order.

4- Acceptance testing: Confirms whether the system meets the business requirements.

Goal: Ensure it's ready for delivery to the customer.

Eg: A client checks if a hotel booking system meets all requested features.

5- Regression testing: Re running previously passed tests after changes.

Goal: Ensures new code hasn't broken existing functionality

Eg. After fixing a bug, testers recheck the app to confirm nothing else broke.

6- Performance testing: Measures system speed, responsiveness, and stability

Goal: Find bottlenecks under different load levels

Eg: Simulating 10,000 users logging in simultaneously

7- Security testing: Identifies vulnerabilities in the system

Goal: Protect data and system integrity

Eg: Testing for SQL injection or weak password policies.

8- Usability testing: Assesses how easy and intuitive the app is for users.

Goal: Improve user experience.

Eg: Observing users navigate an app to see where they get confused.

How Software testing supports Software Quality Assurance

- Bug Detection Early: Testing reveals defects at various stages, preventing bigger issues later.
- Ensures Requirements Are met: Acceptance and system testing ensure the software delivers what was promised.
- Boosts Reliability and Performance: Performance and stress testing shows how stable the system is under pressure.
- Increases User Trust: Usability and security testing lead to safer smoother user experience