OOSE Module 6 - QB Answers

[4 - marks]

1. Define Re-engineering in the context of software maintenance

- Software re-engineering is the process of examining, analyzing, and modifying an existing software system to improve its quality and maintainability. It is mainly used during software maintenance when a system becomes old, complex, or difficult to manage.
- In the context of software maintenance, re-engineering helps improve performance, reliability, and structure without completely rewriting the software. It focuses on improving internal design rather than changing the main functionality.
- Re-engineering includes activities like reverse engineering, code restructuring, and data reconstruction. These activities help developers understand the old system and make it easier to update.
- It also reduces long-term maintenance cost by cleaning poorly written code and updating outdated technologies. Thus, re-engineering is a cost-effective and safer alternative to developing new software from scratch.

2. Explain the main goal of Web Engineering

- The main goal of Web Engineering is to develop high-quality web applications using a systematic and disciplined engineering approach. It ensures that web applications are reliable, scalable, secure, and easy to maintain.
- Web Engineering focuses on handling the complexity of modern web applications that serve a large number of users. These applications need to work continuously without failure and must support different devices and browsers.
- Another important goal is to improve performance and usability. Users expect fast loading pages, smooth navigation, and a good user experience.
- Web Engineering also aims to manage frequent changes in requirements and technologies. By following proper engineering practices, web applications can be updated easily while maintaining quality and security.

3. List the two main components/sides of a Client–Server Software Engineering architecture

- **Client Side**
  The client side is the part of the system that interacts directly with the user. It includes applications such as web browsers, mobile apps, or desktop software. The client sends requests to the server for data, services, or processing.
- The client is mainly responsible for user interface and presentation of information. It does not store large amounts of data or perform complex processing. This makes client devices lightweight and easy to use.

- **Server Side**
  The server side is responsible for processing client requests and managing resources. It performs tasks such as database access, data storage, calculations, and application logic.
- The server sends responses back to the client after completing the requested operations. This separation improves performance, security, and centralized control of data.

4. Explain the concept of a Component in Component-Based Software Engineering

- A component in Component-Based Software Engineering is a self-contained and reusable software unit. It performs a specific function and can be independently deployed in different systems.
- Components interact with other components only through well-defined interfaces. This hides internal implementation details and improves system security and clarity.
- One of the main features of a component is reusability. A component developed once can be reused in multiple applications, reducing development effort and time.
- Components are also replaceable, meaning one component can be swapped with another if the interface remains the same. This improves flexibility and maintainability.
- By using components, software systems become modular, easier to test, and more reliable because many components are pre-tested.

5. State the full form of CASE and its primary function in software development

- The full form of CASE is **Computer-Aided Software Engineering**. CASE refers to the use of computer-based tools to support and automate different activities in software development.
- The primary function of CASE is to improve the quality and productivity of software development. It helps developers, designers, testers, and managers perform their tasks more efficiently.
- CASE tools assist in activities such as requirement analysis, system design, coding, testing, documentation, and project management. By automating repetitive tasks, CASE reduces human errors.
- Another important function of CASE is maintaining proper documentation. It stores diagrams, reports, and design documents in a central repository, which makes software easier to understand and maintain.
- Overall, CASE provides a structured and disciplined approach to software development, ensuring better planning, consistency, and higher-quality software systems.

6. Describe the main difference between Forward Engineering and Reverse Engineering

- **Forward Engineering** is the process of developing software from high-level designs to actual code. It starts with requirements and design and moves step by step towards implementation and deployment.
- Forward Engineering focuses on building new or improved systems using modern tools and technologies. It converts design models into working software systems.
- **Reverse Engineering**, on the other hand, is the process of analyzing an existing software system to understand its design and structure. It works in the opposite direction, moving from code to design.
- Reverse Engineering is mainly used in re-engineering to understand old or poorly documented systems. It helps developers identify business rules and system logic hidden in the code.

7. Name two major advantages of Component-Based Software Engineering

- **Reusability**
  One major advantage of Component-Based Software Engineering (CBSE) is reusability. Components are designed to be reused in multiple applications without modification. This reduces development time and effort.
- Reusing components also improves reliability because reused components are usually well-tested. This leads to fewer bugs and higher software quality.
- **Reduced Development Cost and Time**
  CBSE reduces overall development cost because developers do not need to write code from scratch for every feature. Existing components can be assembled quickly to build systems.
- Faster development also helps organizations meet deadlines and respond quickly to changing requirements. This makes CBSE very useful in modern software development environments.

8. Explain the challenges associated with the Client/Server paradigm

- One major challenge of the Client/Server paradigm is **limited scalability**. As the number of clients increases, the server may become overloaded, reducing system performance.
- Another challenge is **high network dependency**. The system relies heavily on network connectivity, and any network failure can stop communication between clients and servers.
- **Security issues** are also a concern because data is transmitted over networks. Unauthorized access, data breaches, and attacks can occur if security is weak.
- The Client/Server architecture is also **complex to design and maintain**. It involves multiple components like servers, clients, middleware, and protocols, which must work together smoothly.

[5 – marks]

1. Explain the Client/Server Software Engineering paradigm. Discuss the distribution of logic and data between the client and server.

- The Client/Server Software Engineering paradigm is a network-based model where responsibilities are divided between client machines and server machines. The client requests services, and the server processes those requests and sends responses.
- In this paradigm, the **client side** mainly handles the user interface and user interaction. It displays data to the user and collects input, but it performs very little processing or data storage.
- The **server side** is responsible for managing data, application logic, and processing client requests. It stores data in databases, applies business rules, and ensures data consistency and security.
- This separation of logic and data improves performance and maintainability. Clients remain lightweight, while servers handle complex operations centrally.
- By distributing responsibilities in this way, the Client/Server paradigm supports multiple users, improves data control, and makes system updates easier, since most changes are done on the server side.

2. Describe a scenario where Re-engineering becomes necessary for a legacy system. What are the typical steps involved?

- Re-engineering becomes necessary when a legacy system becomes outdated, difficult to maintain, or incompatible with modern hardware and software platforms. For example, an old banking system written in outdated programming languages may not support new security standards or online services.
- In such cases, replacing the entire system may be risky and expensive. Re-engineering allows the organization to improve the existing system while keeping its core functionality intact.
- The typical steps involved in re-engineering begin with **planning**, where goals and scope are defined. This is followed by **analysis**, where the existing system is studied to identify problems.
- Next comes **design**, where improvements are planned, and **implementation**, where code and data are modified. After this, **testing** ensures correctness, and finally **deployment** makes the improved system available to users.
- These steps help modernize the legacy system in a controlled and cost-effective manner.

3. Discuss the key features and characteristics of Web Engineering that distinguish it from traditional software development.

- Web Engineering is different from traditional software development because it focuses on building web-based applications that operate over the internet. These applications are distributed in nature and run across multiple servers and networks.
- One key feature of Web Engineering is **high usability requirements**. Web applications must be easy to use, visually appealing, and responsive across different devices and browsers.
- Another important characteristic is **continuous evolution**. Web applications require frequent updates, feature additions, and security patches due to changing user needs and technologies.
- Web Engineering also emphasizes **performance sensitivity**, as users expect fast load times and smooth interaction. Even small delays can reduce user satisfaction.
- **Security** is a major concern in Web Engineering because web applications are exposed to threats like hacking and data breaches.
- Due to these characteristics, Web Engineering uses iterative development models, strong testing practices, and user-centric design approaches to ensure quality and reliability.

4. Describe how Computer-Aided Software Engineering (CASE) tools can be used to automate the Requirement Elicitation and Modeling phases.

- Computer-Aided Software Engineering (CASE) tools play an important role in automating the requirement elicitation and modeling phases of software development. Requirement elicitation involves collecting, analyzing, and organizing user requirements, which can be complex and time-consuming if done manually.
- CASE tools provide **requirement management tools** that help in gathering requirements in a structured manner. These tools allow developers to document requirements, categorize them, and track changes over time. This reduces confusion and improves clarity.
- During the modeling phase, CASE tools offer **diagramming and analysis tools** such as data flow diagrams, use case diagrams, and system models. These graphical models help in visualizing system behavior and structure.
- CASE tools can automatically check for inconsistencies, missing requirements, and errors in diagrams. By automating these phases, CASE tools improve accuracy, reduce manual effort, and ensure better communication between stakeholders and developers.

5. Explain how CBSE enhances system reliability and reduces time-to-market.

- Component-Based Software Engineering (CBSE) enhances system reliability by using **pre-built and pre-tested software components**. Since these components are already tested in earlier applications, the chances of defects are reduced.
- Reusing reliable components improves overall system stability and reduces unexpected failures. Developers can trust the behavior of components instead of writing new code that may contain errors.
- CBSE also reduces **time-to-market** by speeding up the development process. Instead of developing every feature from scratch, developers can quickly assemble systems using existing components.
- This approach minimizes coding effort and shortens development cycles. Faster development allows organizations to release software earlier and respond quickly to market demands.
- Additionally, modular components make integration and testing easier. As a result, CBSE supports faster delivery, higher reliability, and better quality software systems.

6. Provide two real-world examples of software development activities that can be automated using a CASE tool.

- One real-world example of an activity automated by CASE tools is **diagram creation and system modeling**. CASE tools automatically generate flowcharts, data flow diagrams, ER diagrams, and UML diagrams based on system requirements. This saves time and ensures consistency in design documentation.
- Another example is **documentation generation**. CASE tools can automatically create technical and user documentation from design models and code. This ensures that documentation remains updated and accurate throughout the development process.
- CASE tools can also assist in **code generation**, where basic code structures are generated from design diagrams. This reduces manual coding errors and improves productivity.
- In addition, CASE tools help in **requirement analysis and validation** by detecting missing or conflicting requirements. Automating these activities improves efficiency, reduces errors, and helps developers focus on more complex design and problem-solving tasks.

7. Differentiate between Reverse Engineering and Restructuring as part of the Re-engineering process

- **Reverse Engineering** is the process of analyzing an existing software system to understand its internal structure, design, and functionality. It works in a backward direction, moving from source code to higher-level representations such as design models or documentation.
- The main purpose of reverse engineering is to gain knowledge about a legacy system, especially when proper documentation is missing or outdated. It helps developers rediscover business rules and system logic hidden inside the code.
- **Restructuring**, on the other hand, focuses on improving the internal structure of the software without changing its external behavior. It mainly involves code restructuring, data restructuring, or control flow improvement.
- Restructuring improves readability, maintainability, and performance of the software. Unlike reverse engineering, it does not aim to understand the system but to refine and clean it.
- In short, reverse engineering helps understand *what* the system does, while restructuring improves *how* the system is internally organized.

8. In a Client/Server system, describe the specific roles and responsibilities of the client side and the server side (120+ words)

- In a Client/Server system, the **client side** is responsible for interacting directly with the user. It provides the user interface through which users enter data, view information, and perform actions.
- The client collects user inputs and sends requests to the server using predefined communication protocols. It focuses mainly on presentation logic and does not perform heavy processing or data storage.
- The **server side** handles the core functionality of the system. It processes client requests, applies business logic, and manages access to databases and files.
- The server is responsible for data storage, data security, validation, and consistency. It ensures that multiple clients can access shared resources safely and efficiently.
- After processing the request, the server sends the result back to the client. This clear separation of responsibilities improves system performance, scalability, security, and maintainability in Client/Server architecture

[10 – marks]

1. Analyze Component-Based Software Engineering (CBSE). Discuss its principles, advantages, and the key challenges in successfully integrating third-party components

Component-Based Software Engineering (CBSE)

- Component-Based Software Engineering (CBSE) is a software development approach in which systems are developed by assembling **pre-built and reusable software components**.
- Each component performs a specific task and communicates with other components using **well-defined interfaces**, instead of tightly coupled code.

Principles of CBSE

Reusability : Components are designed to be reused across multiple applications. This reduces repeated development effort and improves efficiency.

Modularity : Each component represents a single logical function. This makes the system easier to understand, develop, and maintain.

Encapsulation : Internal implementation details of a component are hidden. Only interfaces are exposed, improving security and clarity.

Replaceability : A component can be replaced with another if the interface remains the same. This allows flexibility and easier upgrades.

Composability : Components can be combined to form larger and complex systems. This supports scalable and structured development.

Advantages of CBSE

Improved Reliability : Components are pre-tested and widely used, reducing chances of errors.

Reduced Development Time and Cost : Reusing components avoids writing code from scratch.

Better Maintainability : Individual components can be updated without affecting the entire system.

Faster Time-to-Market : Systems can be developed and delivered quickly using existing components.

Challenges in Integrating Third-Party Components

Interface Mismatch : Different components may use incompatible interfaces.

Dependency and Version Conflicts : Components may depend on different versions of libraries.

Lack of Documentation : Poor documentation makes integration difficult.

Security and Trust Issues : Third-party components may contain hidden vulnerabilities.

2. Discuss the different categories of Computer-Aided Software Engineering (CASE) tools. Evaluate their impact on improving software quality and productivity across the software life cycle.

- Computer-Aided Software Engineering (CASE) tools are software tools that support and automate activities across the software development life cycle. They help developers improve productivity and maintain software quality.
- One important category is **Diagramming Tools**, which are used to create flowcharts, data flow diagrams, ER diagrams, and UML diagrams. These tools help visualize system structure and behavior clearly.
- **Requirement Management and Analysis Tools** help in gathering, organizing, and validating requirements. They reduce errors by detecting missing, inconsistent, or conflicting requirements early in development.
- **Design and Modeling Tools** assist in creating system architectures and component models. They ensure standardized design practices and better system planning.
- **Code Generation Tools** automatically generate code from design models. This reduces manual coding effort and minimizes human errors.
- **Documentation Tools** automatically generate technical and user documentation, keeping it consistent and up to date.
- **Central Repository Tools** store all project artifacts in one place, improving coordination and version control.
- The impact of CASE tools is significant. They improve software quality by enforcing standards and reducing defects. Productivity increases because repetitive tasks are automated, allowing developers to focus on problem-solving and innovation. Across the software life cycle, CASE tools ensure consistency, faster development, better documentation, and improved project control.

3. Explain the architecture of Client/Server Software Engineering in detail. Discuss the major architectural styles (2-tier, 3-tier) and their benefits. (180+ words)

- Client/Server Software Engineering architecture is a network-based model in which work is divided between **clients** and **servers**.
- The **client** is the part of the system that interacts with the user. It handles the user interface and sends requests to the server.
- The **server** processes client requests and manages data, databases, and business logic. It sends results back to the client.

Major Architectural Styles

1. Two-Tier Architecture

- In two-tier architecture, the client communicates directly with the server.
- The client handles the user interface and sometimes business logic, while the server manages data storage and processing.

Benefits of Two-Tier Architecture

- Simple to design and implement
- Suitable for small applications
- Faster development due to fewer layers

2. Three-Tier Architecture

- Three-tier architecture introduces an intermediate layer called **middleware or application server**.
- The client handles presentation, middleware handles business logic, and the server manages data.

Benefits of Three-Tier Architecture

- Better scalability for large systems
- Improved security because the client does not directly access the database
- Easier maintenance and updates
- Better performance and load management
- Overall, client/server architecture helps in centralized data control, better performance, and support for multiple users.

4. Describe the complete process of Reengineering a legacy system. Explain the role of Reverse Engineering and Forward Engineering. (180+ words)

- Reengineering is the process of analyzing and modifying an existing software system to improve its quality, performance, and maintainability.
- It is mainly used for **legacy systems** that are old but still important for business operations.

Complete Reengineering Process

1. Planning

- Identify reasons for reengineering such as outdated technology or poor maintainability.
- Define goals, scope, and resources.

2. Analysis

- Study the existing system's code, data, and documentation.
- Identify weaknesses and areas that need improvement.

3. Reverse Engineering

- Reverse Engineering is the process of understanding the existing system by analyzing its source code.
- It converts code into higher-level designs and documentation.
- It helps discover hidden business rules, especially when documentation is missing.

4. Design

- Create an improved design based on analysis results.
- Decide architectural and structural changes.

5. Forward Engineering

- Forward Engineering converts the redesigned models into updated or new code.
- It uses modern tools, languages, and technologies to rebuild the system.

6. Implementation and Testing

- Modify code and data structures.
- Test the system to ensure correctness and performance.

7. Deployment

- Release the reengineered system to users.
- Reverse Engineering helps **understand the old system**, while Forward Engineering helps **rebuild and modernize it effectively**.

5. Challenges in Web Engineering compared to Desktop Applications and Architectural Solutions

Web Engineering faces more complex challenges than desktop application development because web applications operate over the internet and serve a large number of users.

Major Challenges and Solutions :

1. Security Challenges : Web applications are exposed to the internet and can be accessed by anyone. They are vulnerable to attacks such as hacking, SQL injection, cross-site scripting (XSS), and data breaches. Desktop applications usually run locally and face fewer external threats.

Architectural Solutions:

- Use of **multi-tier architecture** to isolate presentation, logic, and data layers.
- Strong **authentication and authorization mechanisms**.
- Use of **encryption, firewalls, and secure communication protocols**.

2. Scalability Challenges : Web applications must support thousands or millions of users simultaneously. Desktop applications usually serve one user at a time.

Architectural Solutions:

- Use of **three-tier and N-tier architectures**.
- **Load balancing** to distribute traffic.
- **Cloud-based deployment** to dynamically scale resources.

3. Performance Challenges : Network delays and heavy traffic can slow response time. Users expect fast and smooth interaction.

Architectural Solutions:

- **Caching mechanisms** to reduce server load.
- **Optimized database design** and efficient server logic.
- Use of **Content Delivery Networks (CDNs)**.

4. Continuous Evolution : Web applications require frequent updates and feature additions.

Architectural Solutions:

- **Modular and component-based architecture**.
- **Agile development methods** to manage changes efficiently

6. Critically evaluate the statement: "CBSE is the future of software development." Support your argument by comparing CBSE with a traditional Object-Oriented Approach.

Critical Evaluation: "CBSE is the Future of Software Development"

- The statement highlights the increasing importance of **Component-Based Software Engineering (CBSE)** in modern software systems.
- CBSE focuses on assembling systems using **pre-built and reusable components**.
- Traditional **Object-Oriented (OO)** development focuses on building software using **classes and objects**, mostly from scratch.

CBSE Approach

- Software is built by combining independent software components. Each component performs a specific function.
- Components interact using **well-defined interfaces**.
- Internal implementation details are hidden from other components.
- This approach supports faster development and easier integration.

Object-Oriented Approach

- Software is developed using classes, objects, and inheritance. Code is mostly written specifically for each application.
- Objects are often tightly coupled with each other.
- Modifying or scaling large OO systems can be difficult.

Comparison Between CBSE and OO

- **Reusability**
    - OO promotes reuse at class level.
    - CBSE promotes reuse of complete functional components. CBSE saves more time and development effort.
- **Development Speed**
    - CBSE enables faster development using existing components.
    - OO development takes more time due to fresh coding.
- **Reliability**
    - CBSE components are usually pre-tested and reliable.
    - OO systems may contain more errors as code is newly written.

7. Choose a specific CASE tool and explain how it aids a software engineer in daily tasks

CASE tool: UML Modeling Tool

Introduction

- A UML modeling tool is a type of **CASE tool** used to create visual models of a software system.
- It helps software engineers design, analyze, and document systems before and during development.

How a UML Modeling Tool Helps in Daily Tasks

- **System Understanding**
    - UML diagrams such as use case, class, and sequence diagrams help engineers clearly understand system requirements.
    - Visual models make complex systems easier to analyze.
- **Requirement Analysis**
    - Use case diagrams help identify user interactions and system functionality.
    - This reduces ambiguity and improves communication with stakeholders.
- **Design Support**
    - Class diagrams help engineers design system structure and relationships.
    - Sequence diagrams show interaction between components over time.
- **Error Reduction**
    - UML tools can automatically check for inconsistencies in diagrams.
    - This helps detect design errors early.
- **Documentation**
    - UML models act as technical documentation.
    - Documentation stays updated as diagrams are modified.
- **Team Collaboration**
    - Shared models help teams communicate ideas clearly.
    - Developers, testers, and managers can understand the same design.

UML modeling tools improve productivity, design quality, and communication.They help engineers plan better and reduce rework during development.

8. Discuss the current trends in Software Engineering required to support cloud and mobile computing

Cloud and mobile computing have changed how software is developed and delivered. Modern software engineering must adopt new trends to meet scalability, performance, and availability demands.

Key Current Trends

1. Cloud-Based Architecture

- Applications are deployed on cloud platforms instead of local servers.
- Cloud supports scalability, availability, and cost efficiency.

2. Microservices Architecture

- Large applications are divided into small, independent services.
- Each service can be developed and deployed separately.
- Improves scalability and maintenance.

3. Component-Based Software Engineering (CBSE)

- Reusable components speed up development.
- Suitable for cloud and mobile apps with frequent updates.

4. Agile and DevOps Practices

- Agile supports rapid development and continuous feedback.
- DevOps integrates development and operations for faster deployment.

5. API-Driven Development

- APIs allow communication between mobile apps, cloud services, and backend systems.
- Enables cross-platform compatibility.

6. Security-Focused Design

- Cloud and mobile apps face higher security risks.
- Strong authentication, encryption, and access control are essential