

Answer Key

Q.1 Multiple Choice Questions

1. c) Requirement Gathering
2. a) Computer Science and Management Science
3. c) Testable
4. c) Ethnography
5. b) To ensure the requirements are complete and consistent
6. a) Requirements Elicitation
7. c) Context Model
8. c) Sequence Diagram
9. b) Behavioral Modeling
10. b) Automating code generation from models
11. b) Low Coupling
12. b) Iterator

Q.2

A) Explanation of Requirements Engineering and its Importance (6 Marks)

- **Definition (1 mark):** Requirements engineering (RE) is the systematic process of discovering, documenting, analyzing, validating, and managing requirements for a software system. It establishes the bridge between real-world needs and the technical specifications of the software.
- **Importance (5 marks):**
 - **Foundation for Success:** RE forms the foundation for successful software development. Errors in requirements lead to costly rework later in the development lifecycle.
 - **Stakeholder Alignment:** It ensures that all stakeholders (customers, users, developers, etc.) have a shared understanding of what the system should do. This reduces misunderstandings and conflicts.
 - **Reduces Development Costs:** By clearly defining the scope and objectives upfront, RE helps to avoid scope creep and unnecessary features, thereby reducing development costs.
 - **Improves Software Quality:** Well-defined requirements lead to better designed, implemented, and tested software, resulting in higher quality and reliability.
 - **Facilitates Project Management:** RE provides a clear roadmap for the project, enabling better planning, scheduling, and resource allocation.
 - **Enables Traceability:** Requirements traceability allows linking requirements to design, code, and test cases, ensuring that all requirements are implemented and verified.

B) Types of Software Requirements Document (SRD) and their Contents (6 Marks)

- **Introduction (1 mark):** The Software Requirements Document (SRD), also known as the Software Requirements Specification (SRS), is a comprehensive description of the intended purpose and environment for software under development. There isn't a single "type," but rather levels of detail and formality depending on the project. We can classify SRDs based on their focus and audience.
- **Types and Contents (5 marks):**

- **User Requirements Document (URD):**
 - **Audience:** Primarily for the customer and end-users.
 - **Content:** Describes the system from the user's perspective, focusing on what the system should do for them. Uses natural language, diagrams, and scenarios. Avoids technical jargon.
 - **Example Contents:** Goals of the system, user characteristics, user stories, use cases, business processes supported.
- **System Requirements Specification (SRS):**
 - **Audience:** Primarily for developers and testers.
 - **Content:** A more detailed and technical description of the system's functionality, performance, and constraints.
 - **Example Contents:** Functional requirements (detailed descriptions of what the system must do), non-functional requirements (performance, security, reliability, etc.), interface requirements, data requirements.
- **Software Requirements Specification (SRS) based on IEEE 830 standard:**
 - **Audience:** Developers, Testers, Project Managers, and Stakeholders
 - **Content:** A structured document following IEEE 830 standard, containing detailed descriptions of software's functionality, performance, constraints, and quality attributes.
 - **Example Contents:** Introduction, Overall Description, Specific Requirements (functional, non-functional, interface), Supporting Information, and Appendices.

Q.3

A) Requirements Elicitation Techniques (6 Marks)

- **Introduction (1 mark):** Requirements elicitation is the process of gathering requirements from stakeholders. Various techniques exist, each with its own strengths and weaknesses.
- **Techniques, Advantages, and Disadvantages (5 marks):**
 - **Interviews:**
 - **Description:** Directly asking stakeholders about their needs and expectations.
 - **Advantages:** Rich information, can uncover hidden requirements, allows for clarification and follow-up questions.
 - **Disadvantages:** Time-consuming, can be difficult to schedule, relies on the interviewer's skills, stakeholders may not always know what they need.
 - **Questionnaires:**
 - **Description:** Distributing a set of questions to a large number of stakeholders.
 - **Advantages:** Efficient for gathering information from many people, cost-effective, allows for anonymous feedback.
 - **Disadvantages:** Limited depth, difficult to ask follow-up questions, response rate may be low, can be difficult to design effective questions.
 - **Brainstorming:**
 - **Description:** A group activity where participants generate ideas freely.
 - **Advantages:** Encourages creativity, can uncover novel requirements, promotes collaboration.
 - **Disadvantages:** Can be dominated by certain individuals, requires skilled facilitation, may generate many irrelevant ideas.

- **Use Cases:**
 - **Description:** Describing the interactions between users and the system to achieve specific goals.
 - **Advantages:** Easy to understand, focuses on user needs, helps to identify functional requirements.
 - **Disadvantages:** May not capture all non-functional requirements, can be time-consuming to create.
- **Ethnography:**
 - **Description:** Observing users in their natural environment to understand their work practices.
 - **Advantages:** Provides a deep understanding of user needs, uncovers hidden requirements, identifies usability issues.
 - **Disadvantages:** Time-consuming, requires specialized skills, can be intrusive, ethical considerations.
- **Prototyping:**
 - **Description:** Building a preliminary version of the system to get feedback from stakeholders.
 - **Advantages:** Helps visualize requirements, identifies usability issues early, facilitates communication.
 - **Disadvantages:** Can be time-consuming and expensive, stakeholders may focus on the prototype's appearance rather than its functionality.

B) Requirements Validation Activities (6 Marks)

- **Introduction (1 mark):** Requirements validation is the process of ensuring that the documented requirements accurately reflect the stakeholders' needs and are complete, consistent, and feasible.
- **Activities (5 marks):**
 - **Inspections (Reviews):**
 - **Description:** A formal process where a team of reviewers examines the requirements document to identify defects, ambiguities, and inconsistencies.
 - **Process:** Involves planning, preparation, review meeting, rework, and follow-up.
 - **Focus:** Checking for completeness, correctness, consistency, clarity, and feasibility.
 - **Prototyping:**
 - **Description:** Developing a working model of the system or a part of it to demonstrate the requirements and get feedback from stakeholders.
 - **Types:** Throwaway prototypes (used for exploration and then discarded), evolutionary prototypes (evolve into the final system).
 - **Benefits:** Helps to visualize requirements, identify usability issues, and clarify misunderstandings.
 - **Test Case Generation:**
 - **Description:** Creating test cases based on the requirements to verify that the system will behave as expected.
 - **Purpose:** To ensure that the requirements are testable and that the system meets the specified criteria.
 - **Benefit:** Identifies ambiguities and inconsistencies in the requirements.
 - **Requirements Traceability Matrix (RTM):**

- **Description:** A document that maps each requirement to its source, design elements, code modules, and test cases.
- **Purpose:** To ensure that all requirements are implemented and verified.
- **Benefit:** Helps to identify missing requirements or gaps in the development process.
- **Acceptance Testing:**
 - **Description:** Testing the system with real users to ensure that it meets their needs and expectations.
 - **Purpose:** To validate that the system is acceptable to the stakeholders.
 - **Benefit:** Provides confidence that the system will be successful in its intended environment.

Q.4 Solve any TWO of the following:

A) System Modeling (6 Marks)

- **What is System Modeling? (2 marks):** System modeling is the process of developing abstract representations of a system, its components, and their relationships. These models can be graphical, textual, or mathematical and are used to understand, analyze, and communicate the system's behavior and structure.
- **Significance in Software Development (2 marks):**
 - **Understanding Complexity:** Models help to manage the complexity of large software systems by providing simplified views.
 - **Communication:** They facilitate communication among stakeholders (developers, customers, users) by providing a common language.
 - **Analysis and Prediction:** Models allow for analyzing the system's behavior, identifying potential problems, and predicting its performance.
 - **Design and Implementation:** They serve as a blueprint for designing and implementing the system.
 - **Documentation:** Models provide valuable documentation that can be used for maintenance and evolution.
- **Different Stakeholders Involved (2 marks):**
 - **Customers:** Provide requirements and validate the model.
 - **Users:** Provide feedback on usability and functionality.
 - **Analysts:** Create and maintain the models.
 - **Designers:** Use the models to design the system architecture and components.
 - **Developers:** Use the models to implement the system.
 - **Testers:** Use the models to create test cases.
 - **Project Managers:** Use the models to plan and track the project.

B) UML Diagrams (6 Marks)

- **Introduction (1 mark):** UML (Unified Modeling Language) is a standardized modeling language used to visualize, specify, construct, and document the artifacts of a software system.
- **Types of UML Diagrams with Purposes and Examples (5 marks):**
 - **Use Case Diagram:**
 - **Purpose:** Describes the interactions between actors (users or external systems) and the system to achieve specific goals.

- **Example:** A use case diagram for an online shopping system might include actors like "Customer" and "Administrator," and use cases like "Browse Products," "Add to Cart," "Checkout," and "Manage Products."
- **Class Diagram:**
 - **Purpose:** Describes the structure of the system by showing classes, their attributes, and relationships (associations, inheritance, aggregation).
 - **Example:** A class diagram for a library system might include classes like "Book," "Author," "Library," and "Member," with attributes like "title," "authorName," "libraryName," and "memberID," and relationships like "Book is written by Author" and "Member borrows Book."
- **Sequence Diagram:**
 - **Purpose:** Shows the interactions between objects in a sequence over time.
 - **Example:** A sequence diagram for placing an order online might show the interactions between the "Customer," "Web Browser," "Order Service," and "Payment Service" objects, with messages like "enterOrderDetails," "validateOrder," "processPayment," and "confirmOrder."
- **Activity Diagram:**
 - **Purpose:** Models the workflow of a process or activity, showing the sequence of actions and decisions.
 - **Example:** An activity diagram for processing a loan application might show activities like "Receive Application," "Check Credit Score," "Approve Loan," and "Disburse Funds," with decision points based on the credit score and loan amount.
- **State Diagram:**
 - **Purpose:** Describes the different states of an object and the transitions between those states in response to events.
 - **Example:** A state diagram for an "Order" object might include states like "Pending," "Processing," "Shipped," and "Delivered," with transitions triggered by events like "Payment Received," "Order Shipped," and "Order Delivered."
- **Component Diagram:**
 - **Purpose:** Shows the organization and relationships among software components.
 - **Example:** A component diagram for e-commerce application showing components like "User Interface," "Order Management," "Payment Gateway" and their dependencies.
- **Deployment Diagram:**
 - **Purpose:** Represents the physical deployment of software components to hardware nodes.
 - **Example:** A deployment diagram showing the web server, application server, and database server and their connections in a cloud environment.

C) Behavioral Modeling (6 Marks)

- **Concept of Behavioral Modeling (2 marks):** Behavioral modeling describes how a system responds to events or stimuli. It focuses on the dynamic aspects of the system, showing how it changes state and interacts with its environment over time. This helps in understanding the system's functionality and identifying potential issues.
- **Examples (2 marks):**

- **State Diagrams:** A state diagram shows the different states an object can be in and the transitions between those states. For example, a "Traffic Light" object might have states like "Red," "Yellow," and "Green," with transitions triggered by timers.
- **Activity Diagrams:** An activity diagram models the workflow of a process. For example, an activity diagram for "Processing an Order" might show the sequence of activities like "Receive Order," "Check Inventory," "Process Payment," and "Ship Order."
- **Relation to System States (2 marks):** Behavioral models, especially state diagrams, explicitly define the possible states of a system or its components. The transitions between these states are triggered by events, representing the system's response to external or internal stimuli. Activity diagrams, while focusing on workflows, also implicitly define states through the completion of activities, leading to different paths or decisions. Thus, behavioral modeling directly relates to understanding and defining the system's possible states and how it moves between them.

Q.5 Solve any TWO of the following:

A) Architectural Styles (6 Marks)

- **Introduction (1 mark):** Architectural styles are reusable solutions to commonly occurring software architecture problems. They provide a vocabulary and set of constraints for designing a system's structure.
- **Different Architectural Styles, Advantages, and Disadvantages (5 marks):**
 - **Layered Architecture:**
 - **Description:** Organizes the system into layers, where each layer provides services to the layer above it and uses services from the layer below it.
 - **Advantages:** Easy to understand and modify, supports incremental development, promotes separation of concerns.
 - **Disadvantages:** Can lead to performance overhead due to multiple layers of indirection, tight coupling between layers if not designed carefully.
 - **Client-Server Architecture:**
 - **Description:** A distributed architecture where clients request services from servers.
 - **Advantages:** Centralized management, scalability, allows for specialized servers.
 - **Disadvantages:** Single point of failure (the server), can be difficult to manage distributed data, security concerns.
 - **Microservices Architecture:**
 - **Description:** Structures an application as a collection of small, autonomous services, modeled around a business domain.
 - **Advantages:** Independent deployment and scaling, technology diversity, fault isolation.
 - **Disadvantages:** Increased complexity due to distributed nature, requires robust infrastructure, challenging to manage transactions across services.
 - **Pipe and Filter Architecture:**
 - **Description:** Processes data in a series of steps connected by pipes. Each filter performs a specific transformation on the data.
 - **Advantages:** Easy to understand, supports reuse of filters, allows for parallel processing.
 - **Disadvantages:** Can be inefficient for complex transformations, data format must be consistent across filters.

B) Architectural Design Patterns (6 Marks)

- **What are Architectural Design Patterns? (2 marks):** Architectural design patterns are well-documented, reusable solutions to recurring architectural problems in software design. They provide a proven template for structuring a system to achieve specific quality attributes.
- **Role in Creating Robust and Scalable Systems (2 marks):**
 - **Robustness:** Architectural patterns help create robust systems by providing proven solutions for handling errors, ensuring fault tolerance, and managing complexity.
 - **Scalability:** They enable scalability by providing patterns for distributing workload, managing resources, and handling increasing user demand.
- **Examples (2 marks):**
 - **Model-View-Controller (MVC):** Separates the application into three interconnected parts: the model (data), the view (user interface), and the controller (logic). This promotes maintainability and testability.
 - **Layered Pattern:** Organizes the system into layers, each providing a specific set of services. This promotes modularity and separation of concerns.
 - **Microkernel Pattern:** Separates the system into a core kernel and a set of plug-in modules. This allows for flexibility and extensibility.

C) Importance of Architectural Documentation and Different Views (6 Marks)

- **Importance of Architectural Documentation (2 marks):**
 - **Communication:** Architectural documentation serves as a communication tool among stakeholders, including developers, architects, testers, and customers.
 - **Understanding:** It provides a clear understanding of the system's structure, components, and relationships.
 - **Maintenance:** It facilitates maintenance and evolution by providing a roadmap for making changes to the system.
 - **Reuse:** It enables reuse of architectural knowledge and components in future projects.
- **Different Views Used to Represent the Architecture (4 marks):**
 - **Logical View:** Describes the system's functional requirements, including classes, objects, and relationships. Often represented using UML class diagrams and interaction diagrams.
 - **Development View:** Describes the system's module structure, including components, interfaces, and dependencies. Focuses on the organization of the code and the development environment. Often represented using component diagrams.
 - **Process View:** Describes the system's runtime behavior, including processes, threads, and communication mechanisms. Focuses on concurrency, performance, and scalability. Often represented using activity diagrams and sequence diagrams.
 - **Physical View:** Describes the system's deployment topology, including hardware nodes, network connections, and software components. Focuses on the physical infrastructure and deployment environment. Often represented using deployment diagrams.
 - **Use Case View:** Describes the system from the user's perspective, showing how users interact with the system to achieve specific goals. It ties all other views together.

Q.6 Solve any TWO of the following:

A) Categories of Design Patterns (6 Marks)

- **Introduction (1 mark):** Design patterns are categorized into three main groups based on their purpose: creational, structural, and behavioral.
- **Categories and Examples (5 marks):**
 - **Creational Patterns:**
 - **Purpose:** Deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
 - **Examples:**
 - **Singleton:** Ensures that only one instance of a class is created and provides a global point of access to it.
 - **Factory Method:** Defines an interface for creating an object, but lets subclasses decide which class to instantiate.
 - **Abstract Factory:** Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
 - **Structural Patterns:**
 - **Purpose:** Deal with the composition of classes or objects to form larger structures.
 - **Examples:**
 - **Adapter:** Converts the interface of a class into another interface clients expect.
 - **Composite:** Composes objects into tree structures to represent part-whole hierarchies.
 - **Decorator:** Dynamically adds responsibilities to an object.
 - **Behavioral Patterns:**
 - **Purpose:** Deal with algorithms and the assignment of responsibilities between objects.
 - **Examples:**
 - **Observer:** Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
 - **Strategy:** Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
 - **Template Method:** Defines the skeleton of an algorithm in a method, deferring some steps to subclasses.

B) Benefits and Drawbacks of Using Design Patterns (6 Marks)

- **Introduction (1 mark):** Design patterns offer several advantages in software development, but also have potential drawbacks.
- **Benefits (3 marks):**
 - **Reusability:** Design patterns provide proven solutions to common design problems, allowing developers to reuse them in different projects.
 - **Maintainability:** They promote modularity and separation of concerns, making it easier to maintain and modify the code.
 - **Flexibility:** They allow for easy extension and modification of the system's behavior without affecting existing code.

- **Improved Communication:** Design patterns provide a common vocabulary for developers, facilitating communication and understanding.
- **Reduced Development Time:** By reusing proven solutions, developers can reduce the time and effort required to design and implement software.
- **Potential Drawbacks (2 marks):**
 - **Over-Engineering:** Applying patterns unnecessarily can lead to over-complicated designs.
 - **Learning Curve:** Developers need to understand the patterns and their appropriate usage.
 - **Abstraction Overhead:** Introducing abstractions can sometimes lead to performance overhead.
 - **Complexity:** While patterns solve certain problems, they can also add complexity to the code if not used carefully.

C) Application of Design Patterns in Real-World Software Systems (6 Marks)

- **Introduction (1 mark):** Design patterns are widely used in real-world software systems to solve common design problems and improve software quality.
- **Specific Examples (5 marks):**
 - **E-commerce Systems:**
 - **Factory Pattern:** Used to create different types of products (e.g., books, electronics, clothing) without specifying their concrete classes.
 - **Strategy Pattern:** Used to implement different payment methods (e.g., credit card, PayPal, bank transfer) in a flexible and interchangeable way.
 - **Observer Pattern:** Used to notify users about order status updates (e.g., order confirmation, shipping updates, delivery confirmation).
 - **GUI Frameworks:**
 - **MVC Pattern:** Used to separate the user interface (view), data (model), and control logic (controller), making it easier to maintain and test the code.
 - **Composite Pattern:** Used to create complex GUI components (e.g., trees, menus, toolbars) from simpler components.
 - **Decorator Pattern:** Used to add features to existing GUI components dynamically.
 - **Operating Systems:**
 - **Singleton Pattern:** Used to ensure that only one instance of certain system resources (e.g., the file system manager, the window manager) is created.
 - **Command Pattern:** Used to implement undo/redo functionality.
 - **Compiler Design:**
 - **Interpreter Pattern:** Used to interpret programming language statements.
 - **Flyweight Pattern:** Used to reduce memory footprint of similar objects. For example, a character object that's used many times.