Course Code: U/ID 46513/UCC5A

Course Title: Software Engineering

Examination: June 2024

Time: Three hours
Maximum: 75 marks

SECTION A — (10 × 2 = 20 marks)

Answer any TEN questions.

1. What is Software Engineering?
2. Define the term-Managerial issue.
3. What is Planning Activity?
4. What is meant by Maintenance cost?
5. What is Hierarchical Team Structure?
6. What is a Decision Table?
7. Define the term–Data Flow Diagram.
8. What is a Typeless language?
9. What is a Test Plan?
10. What is Verification?
11. Define Quality Assurance.
12. Define Acceptance Testing.

SECTION B — (5 × 5 = 25 marks)

Answer any FIVE questions.

13. Discuss the size categories for software products.
14. Explain the format of a software requirements specification.
15. Discuss structure charts and HIPO diagrams.
16. Write a note on symbolic execution techniques.
17. Explain the concept of Software Design.
18. Explain the basic principles of effective modular design.
19. Illustrate the system testing strategy.

SECTION C — (3 × 10 = 30 marks)

Answer any THREE questions.

20. Discuss briefly the design notation and design techniques.
21. Explain the factors that influence the quality and productivity of software products.
22. Explain any two Software Cost Estimation Techniques.
23. Discuss in detail structured coding techniques.

24. Explain any one source code metric in software maintenance.

Answers

SECTION A

1. **What is Software Engineering?**

   Answer: Software engineering is the systematic application of engineering approaches to the development of software. It involves the use of methodologies, tools, and techniques to design, develop, maintain, and test software systems to ensure they are reliable, efficient, and meet user requirements.

2. **Define the term-Managerial issue.**

   Answer: Managerial issues in software engineering refer to challenges related to planning, organizing, staffing, directing, and controlling the software development process. This includes managing project timelines, budgets, resource allocation, risk management, and ensuring effective communication among team members.

3. **What is Planning Activity?**

   Answer: Planning activity in software engineering involves defining the objectives, scope, resources, schedule, and tasks for a software project. It includes creating project plans, timelines, resource plans, and budget estimates to ensure the project is completed successfully and meets its goals.

4. **What is meant by Maintenance cost?**

   Answer: Maintenance cost refers to the expenses incurred in updating, modifying, and supporting software after its initial deployment. This includes fixing bugs, adding new features, improving performance, and ensuring compatibility with new hardware or software environments.

5. **What is Hierarchical Team Structure?**

   Answer: A hierarchical team structure in software engineering is an organizational model where team members are arranged in levels of authority. The structure includes roles such as project managers, team leads, and developers, with each level having specific responsibilities and reporting relationships.

6. **What is a Decision Table?**

   Answer: A decision table is a tabular method for representing complex decision logic. It lists conditions and corresponding actions, helping to systematically capture and analyze all possible scenarios and their outcomes. It is used for decision-making in software development.

7. **Define the term–Data Flow Diagram.**

Answer: A Data Flow Diagram (DFD) is a graphical representation of the flow of data through a system. It illustrates how data is processed, stored, and transferred between different components of the system, helping to understand the system’s functionality and data movement.

8. **What is a Typeless language?**

Answer: A typeless language, or dynamically typed language, is a programming language where variable types are determined at runtime rather than at compile time. This allows for more flexibility but can lead to runtime errors if type assumptions are incorrect.

9. **What is a Test Plan?**

Answer: A test plan is a document outlining the strategy, objectives, resources, schedule, and scope of testing activities for a software project. It defines what will be tested, the testing criteria, test environments, test deliverables, and roles and responsibilities.

10. **What is Verification?**

Answer: Verification in software engineering is the process of evaluating work products to ensure they meet specified requirements and standards. It involves reviews, inspections, and testing to confirm that the software correctly implements the intended design.

11. **Define Quality Assurance.**

Answer: Quality Assurance (QA) is a systematic process to ensure that a software product meets specified quality standards and requirements. It involves planned and systematic activities, such as process monitoring, audits, and testing, to prevent defects and ensure quality.

12. **Define Acceptance Testing.**

Answer: Acceptance testing is the final phase of testing before the software is delivered to the customer. It verifies that the software meets the business requirements and is ready for use. It involves testing by the end-users or clients to ensure the system functions as expected.

SECTION B

13. **Discuss the size categories for software products.**

Answer: Software products are often categorized by size to estimate effort, cost, and resources needed for development. Categories can include:
- **Small projects:** Typically involve less than 25,000 lines of code, are developed by small teams, and have a short development cycle.
- **Medium projects:** Range between 25,000 and 100,000 lines of code, require larger teams, and have moderate complexity and duration.
- **Large projects:** Exceed 100,000 lines of code, involve large teams,

extended timelines, and significant complexity in terms of functionality and integration.

14. **Explain the format of a software requirements specification.**

Answer: A Software Requirements Specification (SRS) document typically includes the following sections:
- **Introduction:** Purpose, scope, definitions, acronyms, references, and overview.
- **Overall Description:** Product perspective, product functions, user characteristics, constraints, assumptions, and dependencies.
- **Specific Requirements:** Detailed functional requirements, data requirements, interface requirements, and performance requirements.
- **Appendices:** Glossary, document references, and other supplementary information.

15. **Discuss structure charts and HIPO diagrams.**

Answer:
-- **Structure Charts:**
1. Hierarchical representation of system architecture.
2. Clear visualization of modules and their relationships.
3. Promotes modular design and scalability.

- **HIPO Diagrams:**
1. Combines hierarchical structure with input-process-output details.
2. Depicts system architecture and detailed data flow.
3. Effective for communication, documentation, and analysis.

16. **Write a note on symbolic execution techniques.**

Answer:

Symbolic Execution Techniques:

1. **Representation of Inputs:**
- Symbolic values represent program inputs instead of concrete data.

2. **Path Exploration:**
- Systematically traverses program paths by solving symbolic expressions.

3. **Error Identification:**
- Helps pinpoint potential errors and vulnerabilities in the codebase.

4. **Security Analysis:**
- Crucial for detecting security vulnerabilities like injection attacks or authentication bypasses.

5. **Code Coverage Improvement:**
- Facilitates the generation of test cases for achieving comprehensive code coverage.

6. **Complexity Challenges:**
- Faces scalability issues due to path explosion in complex programs.

7. **Integration with Analysis Techniques:**
- Often integrated with static analysis approaches for enhanced effectiveness.

17. **Explain the concept of Software Design.**

Answer:

Concept of Software Design:

1. **Requirement Satisfaction:**
- Aims to fulfill specified requirements through systematic planning and structuring.

2. **Architecture Definition:**
- Involves outlining the high-level structure and organization of the software system.

3. **Component Specification:**
- Identifying individual components and their functionalities within the system.

4. **Interface Design:**
- Establishing clear communication and interaction points between system components.

5. **Data Management:**
- Designing the structure and handling of data throughout the software lifecycle.

6. **Abstraction and Modularity:**
- Emphasizes abstraction to manage complexity and modularity for ease of maintenance and scalability.

7. **Iterative Process:**
- Often iterative, allowing for refinement and optimization as the project progresses.

18. **Explain the basic principles of effective modular design.**

Answer: Effective modular design follows principles such as:
- **Cohesion:** Modules should have a single, well-defined purpose.
- **Coupling:** Minimize dependencies between modules.
- **Abstraction:** Hide the internal details of modules from each other.
- **Encapsulation:** Keep data and behavior within modules private.
- **Separation of Concerns:** Divide the system into distinct features with minimal overlap.
- **Reusability:** Design modules that can be reused in different contexts.

19. **Illustrate the system testing strategy.**

Answer: System testing strategy involves:

- **Planning:** Define objectives, scope, test cases, and environment.
- **Test Case Design:** Develop test cases covering functional and non-functional requirements.
- **Environment Setup:** Prepare the testing environment and data.
- **Execution:** Run tests, record results, and identify defects.
- **Reporting:** Document and report test outcomes and issues.
- **Retesting and Regression Testing:** Fix defects and retest; ensure new changes do not introduce new issues.

SECTION C

20. **Discuss briefly the design notation and design techniques.**

Answer:

Design Notation and Techniques:

1. **UML (Unified Modeling Language):**
   - Provides a standardized notation for visualizing system architecture and behavior.

2. **ERD (Entity-Relationship Diagram):**
   - Illustrates the relationships between entities in a database system.

3. **DFD (Data Flow Diagram):**
   - Offers a graphical representation of data movement and processing within a system.

4. **Flowcharts:**
   - Represents the sequence of operations or steps in a process, aiding in understanding system workflows.

5. **Modular Design:**
   - Divides a system into smaller, independent modules to facilitate development and maintenance.

6. **CRC Cards (Class-Responsibility-Collaboration):**
   - Used in object-oriented design to define classes, their responsibilities, and collaborations.

7. **Design Patterns:**
   - Reusable solutions to common design problems, promoting best practices and maintainability.

8. **Prototype Design:**
   - Involves creating a preliminary version of the system to gather feedback and validate design decisions.

9. **User-Centered Design:**
   - Focuses on understanding user needs and preferences to create intuitive and user-friendly interfaces.

10. **Agile Design:**
- Adapts design processes to accommodate changing requirements and iterative development cycles, fostering flexibility and responsiveness.

21. **Explain the factors that influence the quality and productivity of software products.**

Answer:

Factors Influencing Software Quality and Productivity:

1. **Testing and Quality Assurance:**
- Comprehensive testing strategies and quality assurance processes ensure the reliability and correctness of the software.

2. **Documentation:**
- Thorough and well-maintained documentation aids in understanding, maintaining, and extending the software system.

3. **User Feedback and Iterative Improvement:**
- Incorporating user feedback and iteratively improving the software enhances its usability and relevance.

4. **Resource Allocation:**
- Adequate allocation of resources including time, budget, and personnel influences the software's development trajectory.

5. **Technology Stack:**
- Choice of appropriate technologies, frameworks, and libraries impacts both the efficiency of development and the performance of the final product.

6. **Regulatory Compliance:**
- Adherence to industry standards, regulations, and compliance requirements ensures legal and ethical integrity.

7. **Risk Management:**
- Identification, assessment, and mitigation of potential risks throughout the software development lifecycle contribute to smoother project execution.

8. **Customer Engagement:**
- Engaging customers and stakeholders throughout the development process fosters alignment with their needs and expectations.

9. **Feedback Loops:**
- Establishing feedback loops within the development team and with stakeholders enables continuous improvement and adaptation.

10. **Code Maintainability:**
- Writing clean, modular, and maintainable code reduces technical

debt and facilitates future enhancements or bug fixes.

22. **Explain any two Software Cost Estimation Techniques.**

Answer:

Software Cost Estimation Techniques:

1. **COCOMO (Constructive Cost Model):**
   - **Parameter-based Estimation:** COCOMO estimates cost based on parameters such as project size, complexity, and cost drivers like team experience and development tools.
   - **Three Variants:** It offers three variants - Basic, Intermediate, and Detailed, each suitable for different project stages and complexities.
   - **Mathematical Model:** Utilizes mathematical formulas and historical data to predict effort, schedule, and cost accurately.
   - **Risk Management:** COCOMO incorporates risk management by considering uncertainty factors and providing sensitivity analysis.
   - **Industry Standard:** Widely used in the industry due to its versatility and ability to tailor estimates to specific project characteristics.

2. **Function Point Analysis (FPA):**
   - **Functional Measurement:** FPA measures software size based on functional components like inputs, outputs, inquiries, and files.
   - **Technology-agnostic:** It is technology-agnostic, focusing solely on the functionality provided to the user, making it applicable across various development platforms.
   - **Complexity Adjustment:** Accounts for differences in complexity by assigning weights to different types of functions.
   - **Effort Estimation:** Estimates effort and cost based on the calculated function points and productivity metrics.
   - **Objective Measurement:** Offers an objective measurement of software size, facilitating more accurate cost estimation compared to other techniques.

23. **Discuss in detail structured coding techniques.**

Answer:

Structured Coding Techniques:

1. **Information Hiding:**
   - Encapsulation of implementation details within modules to minimize dependencies and promote modular design.

2. **Data Abstraction:**
   - Representing data objects with well-defined interfaces, abstracting away implementation details to improve code readability and maintainability.

3. **Single Entry, Single Exit (SESE):**
   - Structuring code blocks to have a single entry and a single exit point, enhancing code clarity and easing debugging.

4. **Code Reusability:**
- Designing modules and functions to be reusable across different parts of the program or in future projects, reducing redundancy and promoting efficiency.

5. **Structured Error Handling:**
- Using structured mechanisms such as try-catch blocks or exception handling to manage errors and exceptions gracefully, improving program reliability.

6. **Standardization:**
- Adhering to coding standards and best practices, such as those defined by organizations like IEEE or ISO, to ensure consistency and interoperability among codebases.

7. **Testing and Debugging Facilitation:**
- Structuring code in a modular and predictable manner aids in testing and debugging efforts, as individual components can be isolated and tested independently.

8. **Scalability:**
- Structured coding techniques facilitate scalability by allowing the system to grow and evolve without sacrificing maintainability or introducing complexity.

9. **Performance Optimization:**
- By organizing code logically and efficiently, structured coding techniques can lead to better performance through optimized algorithms and data structures.

10. **Version Control Integration:**
- Structured coding practices align well with version control systems, enabling better collaboration and tracking changes across the development lifecycle.

24. **Explain any one source code metric in software maintenance.**

Answer:

Cyclomatic Complexity:

1. **Quantifying Code Complexity:**
- Measures the number of linearly independent paths through a program's source code, indicating its structural complexity.

2. **Calculation Method:**
- Calculated using the formula $M = E - N + 2P$, where M is the cyclomatic complexity, E is the number of edges, N is the number of nodes, and P is the number of connected components.

3. **Identifying Potential Trouble Spots:**
- Higher cyclomatic complexity values suggest the presence of more

decision points and branching in the code, indicating areas that may be prone to errors or difficult to maintain.

4. **Guiding Testing Efforts:**
   - Helps prioritize testing efforts by focusing on complex modules with higher cyclomatic complexity values, as they are more likely to contain defects.

5. **Maintenance Impact:**
   - Modules with high cyclomatic complexity are harder to understand, modify, and maintain, potentially leading to increased maintenance costs and effort.

6. **Thresholds and Guidelines:**
   - Establishing thresholds for cyclomatic complexity helps teams identify code that may need refactoring or optimization. Commonly, a complexity of 10 or lower is considered manageable.

7. **Refactoring Opportunities:**
   - Identifies opportunities for refactoring by breaking down complex code into simpler, more modular components, improving readability and maintainability.

8. **Tool Support:**
   - Many software development tools and IDEs provide built-in support for calculating cyclomatic complexity, making it easier to assess code quality during development and maintenance.

9. **Cross-language Applicability:**
   - Applicable across various programming languages, allowing for consistent evaluation of code complexity regardless of the language used.

10. **Educational Value:**
   - Helps developers understand the structural complexity of their code and encourages the adoption of best practices for writing more maintainable and efficient software.
