**SRM Institute of Science and Technology**

**SET - C**

**College of Engineering and Technology**

**School of Computing**

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

**Academic Year: 2024-2025 (EVEN)**

**Answer Key**

**21CSC303J - Software Engineering and Project Management (FJ3 Test)**

**Part A: Multiple Choice Questions (10 × 1 = 10 Marks)**

1. **Risk strategy that focuses on \_\_\_\_\_\_\_\_–– \_\_\_\_\_\_\_\_ preventive actions before the risk occurs:**  
   **Answer**: B) Proactive –– Taking actions early  
   **Explanation**: Proactive risk strategies involve anticipating and addressing risks before they occur, unlike reactive strategies that respond after the risk manifests.
2. **In RMMM, match correctly: M1 – Management, M2 – Mitigation, M3 – Monitoring. What does the third 'M' represent?**  
   **Answer**: C) M3  
   **Explanation**: RMMM stands for Risk Management, Mitigation, and Monitoring. The third 'M' refers to Monitoring, which involves tracking risks throughout the project.
3. **Match the activities with their relevance to Product Release Management. Which matching is correct?**  
   **Correct Answer**: A) A-1, B-2, C-3, D-4  
   **Explanation**: Product release management involves planning (1), testing (2), versioning (3), and deployment (4), assuming the options align with these activities.
4. **Reengineering is typically applied when:**  
   **Answer**: C) An old product needs structural or performance improvement  
   **Explanation**: Reengineering is used to improve the structure, performance, or maintainability of existing software, not for new builds or bug-free systems.
5. **Risk refinement in RMMM involves:**  
   **Answer**: B) Breaking down identified risks into manageable components  
   **Explanation**: Risk refinement breaks down complex risks into smaller, actionable components for better management and mitigation.
6. **Match each element with its category. Select the correct match:**  
   **Correct Answer**: A) A-1, B-2, C-3, D-4  
   **Explanation**: Unit testing (1), integration testing (2), validation testing (3), and system testing (4) align with their respective categories.
7. **Deskchecks are used mainly for \_\_\_\_\_\_\_ –– \_\_\_\_\_\_\_\_\_ the correctness of logic manually.**  
   **Answer**: B) Verifying –– Ensuring  
   **Explanation**: Deskchecks involve manually reviewing code to verify and ensure the correctness of logic, not automating or debugging.
8. **Identify the true black-box testing technique:**  
   **Answer**: B) Boundary value analysis  
   **Explanation**: Boundary value analysis is a black-box testing technique that tests input boundaries, unlike path coverage, statement coverage, or loop testing, which are white-box techniques.
9. **Assertion (A): Integration testing focuses on interaction between modules. Reason (R): It ensures modules individually work without bugs.**  
   **Answer**: C) A is true but R is false  
   **Explanation**: Integration testing focuses on module interactions (A is true), but it does not ensure individual module functionality, which is the role of unit testing (R is false).
10. **"Walkthrough" in software engineering is best described as:**  
    **Answer**: A) Informal peer review presented by the author  
    **Explanation**: A walkthrough is an informal review led by the code’s author to identify issues, unlike formal inspections or automated testing.

**Part B: Short Answer Questions (4 × 5 = 20 Marks)**

1. **How do coding frameworks naturally support consistency and reusability, and where might their advantages become most visible?**  
   **Answer**:  
   Coding frameworks (e.g., Django, Spring) provide predefined structures, libraries, and conventions that enforce consistent coding practices across teams. They support reusability through modular components, such as reusable libraries or templates, reducing redundant code. Advantages are most visible in large-scale projects with multiple developers, where frameworks streamline development, ensure uniform code quality, and accelerate feature implementation (e.g., MVC architecture in web apps). They also simplify maintenance by standardizing updates.  
   **Key Points**:

* Consistency via standardized patterns.
* Reusability through modules/libraries.
* Visible in collaborative, scalable projects.

1. **How do black-box and white-box testing differ, and what considerations guide their use?**  
   **Answer**:  
   Black-box testing focuses on testing functionality without knowledge of internal code, using techniques like boundary value analysis or equivalence partitioning. White-box testing examines internal logic paths, using techniques like statement or path coverage. Black-box is ideal for validating user requirements, while white-box ensures code correctness. Considerations include project stage (black-box for acceptance, white-box for unit testing), tester expertise, and resource availability.  
   **Key Points**:

* Black-box: External functionality, user-focused.
* White-box: Internal logic, developer-focused.
* Choice depends on testing goals and resources.

1. **How might a company balance reactive and proactive risk strategies, and what real-world examples highlight their strengths and weaknesses?**  
   **Answer**:  
   Proactive strategies anticipate risks (e.g., regular security audits), while reactive strategies address issues post-occurrence (e.g., patching after a breach). Balancing involves prioritizing proactive measures for high-impact risks and maintaining reactive plans for unforeseen issues. Example: Proactive antivirus updates prevent malware (strength: prevention) but may miss new threats (weakness). Reactive incident response after a data breach (strength: quick recovery) risks data loss (weakness).  
   **Key Points**:

* Proactive: Prevent risks, cost-effective.
* Reactive: Quick fixes, but costly if frequent.
* Balance via risk assessment and contingency plans.

1. **Your self-driving car project faces sensor failure risks—how does an RMMM plan keep it on track?**  
   **Answer**:  
   An RMMM (Risk Management, Mitigation, Monitoring) plan identifies sensor failure risks, assesses their impact (e.g., safety hazards), and prioritizes them. Mitigation includes redundant sensors or fault-tolerant algorithms. Monitoring involves real-time diagnostics to detect failures. Management assigns roles, budgets, and timelines for mitigation. Regular testing ensures reliability, keeping the project on track by minimizing downtime and ensuring safety compliance.  
   **Key Points**:

* Identify and prioritize sensor risks.
* Mitigate with redundancy and diagnostics.
* Monitor and manage for continuous reliability.

1. **Why does software maintenance quietly become central to sustaining a product’s value throughout its lifecycle?**  
   **Answer**:  
   Software maintenance ensures a product remains functional, secure, and relevant post-launch. It addresses evolving user needs, fixes bugs, enhances performance, and adapts to new technologies or regulations. Maintenance sustains value by preventing obsolescence and ensuring user satisfaction. For example, regular updates to a banking app maintain security and usability, retaining customers.  
   **Key Points**:

* Fixes bugs, improves performance.
* Adapts to user and tech changes.
* Sustains long-term product relevance.

**Part C: Long Answer Questions (2 × 10 = 20 Marks)**

1. **Scenario: A team is developing a banking software and follows a structured testing strategy. Describe how they would implement each testing level: unit, integration, validation, and system. Highlight the key purpose of each.**  
   **Answer**:

* **Unit Testing**: Test individual components (e.g., a function calculating interest rates) in isolation using tools like JUnit. Developers write test cases to verify logic correctness. **Purpose**: Ensure each module works as intended.
* **Integration Testing**: Test interactions between modules (e.g., data transfer between account management and transaction processing). Use top-down or bottom-up approaches. **Purpose**: Verify modules work together without errors.
* **Validation Testing**: Test the software against user requirements (e.g., ensuring the UI allows secure fund transfers). Conducted via black-box techniques like user acceptance testing. **Purpose**: Confirm the software meets user expectations.
* **System Testing**: Test the entire system as a whole (e.g., end-to-end banking workflows). Includes performance, security, and stress testing. **Purpose**: Ensure the integrated system is reliable, secure, and functional.  
  **Implementation**: Use automated tools (e.g., Selenium for system testing) and manual reviews, with test cases derived from requirements.  
  **Key Points**: Each level builds on the previous, ensuring comprehensive quality.

1. **(OR) Scenario: A code walkthrough is conducted, and multiple logical issues are identified. Explain the process, stakeholders, and how it differs from inspections and deskchecks. Discuss how such reviews contribute to quality assurance.**  
   **Answer**:

* **Walkthrough Process**: The author presents the code to a small team, explaining logic and functionality. Peers review for logical errors, inconsistencies, or improvements, discussing issues informally. Issues are documented for fixes.
* **Stakeholders**: Author, peer developers, and sometimes a moderator or tester.
* **Differences**:
  + **Inspections**: Formal, structured reviews with predefined roles (e.g., inspector) and checklists, focusing on defect detection.
  + **Deskchecks**: Individual, manual code review by the author or a single reviewer, without group discussion.
  + **Walkthroughs**: Informal, collaborative, and author-led, emphasizing understanding and feedback.
* **Quality Assurance Contribution**: Walkthroughs catch logical errors early, improve code clarity, and foster team knowledge sharing. They reduce defects before testing, saving time and costs.  
  **Key Points**: Informal, collaborative reviews enhance code quality and team synergy.

1. **Scenario: A company is preparing to release a new version of its software product. Discuss the steps involved in product release management, including planning, testing, versioning, and deployment. How can risk management be integrated into this process?**  
   **Answer**:

* **Steps in Product Release Management**:
  1. **Planning**: Define release goals, scope, and timeline. Identify features and assign tasks (e.g., using Agile sprints).
  2. **Testing**: Conduct unit, integration, system, and acceptance testing to ensure functionality, performance, and security. Use tools like Jenkins for automation.
  3. **Versioning**: Assign version numbers (e.g., 2.1.0) using semantic versioning to track changes and ensure compatibility.
  4. **Deployment**: Roll out the software via staged releases (e.g., beta, production) using CI/CD pipelines or manual deployment.
* **Risk Management Integration**:
  1. **Identify Risks**: E.g., deployment failures, compatibility issues.
  2. **Mitigation**: Use rollback plans, test in staging environments, and maintain backups.
  3. **Monitoring**: Track release performance with logs and user feedback.
  4. **Management**: Assign risk owners and contingency budgets.  
     **Example**: A failed deployment can be mitigated with a rollback to the previous version.  
     **Key Points**: Structured release management with integrated RMMM ensures reliable, low-risk releases.

1. **(OR) Scenario: An outdated HR management system is facing performance issues. The company has decided to re-engineer the software. Explain the process of software reengineering, its benefits, and the key challenges faced during this transformation.**  
   **Answer**:

* **Software Reengineering Process**:
  1. **Inventory Analysis**: Assess the existing HR system’s codebase, identifying outdated components.
  2. **Reverse Engineering**: Extract design and requirements from legacy code to understand functionality.
  3. **Restructuring**: Refactor code to improve structure (e.g., modularize monolithic code).
  4. **Forward Engineering**: Rebuild the system with modern technologies (e.g., cloud-based architecture).
  5. **Testing and Migration**: Test the reengineered system and migrate data from the old system.
* **Benefits**:
  1. Improved performance and scalability.
  2. Enhanced maintainability and reduced technical debt.
  3. Better user experience with modern features.
* **Challenges**:
  1. **Complexity**: Understanding poorly documented legacy code.
  2. **Cost**: High resource and time investment.
  3. **Data Migration**: Risk of data loss or corruption.
  4. **User Resistance**: Employees may resist new workflows.  
     **Key Points**: Reengineering modernizes systems but requires careful planning to overcome challenges.