SANGAM UNIVERSITY

File Type: ASSIGNMENT ☑ PROJECT ☐ PRACTICAL ☐ CASE STUDY ☐

NAME: Harshvardhan Randhawa

ENROLLMENT NO: 2021BTCS008

SESSION: 2045-25 COURSE: B.TECH

BRANCH: CSE SEMESTER: VII

SUBJECT: Software Testing & Quality Assurance

SUBJECT CODE: PEC-CS-13

GUIDED BY: Prof. Raj Kumar Somani

Verified / Checked

Signature

SANGAM UNIVERSITY
where Aspiration meets Opportunity

1. What is CMM, and how does it provide a framework for improving software development process and software quality?

Soln * The Capability Maturity Model (CMM) is a framework developed by the Software Engineering Institute (SEI) to assess and improve the software development processes of organizations. It defines structured level of process maturity, enabling organizations to gradually improve their process and achieve higher software quality.

* How does CMM provide a framework for improving s/w development process and software quality?

1. Maturity Levels:
   - Level 1 (Initial): Process are adhoc & chaotic
   - Level 2 (Repeatable): Basic project management process are established
   - Level 3 (Defined): Process are standardized & documented
   - Level 4 (Managed): Processes are measured and controlled
   - Level 5 (Optimizing): Focus on continuous process improve.

2. Process Improvement:

Organisations progress through the levels by identifying weakness, implementing best practices and refining their process.

3. Key Process Area (KPAs):

Each level (except 1) has specific KPAs, such as configuration management, quality assurance, and defect prevention, which organizations must address to advance.

4. Focus on Quality:

- Higher level introduce metrics for process control and defect reduction
- Continuous feedback loops ensures ongoing quality improvement.

5. Benchmarking:

Provides a standard framework to compare organization maturity and capability.

Q2. What is Total Quality Management (TQM), and how can it be applied in the context of software development to improve quality across the organization?

Soln TQM, is a management approach focused on embedding quality into every aspect of an aspect of an organization's processes, products and services. It emphasizes continuous improvement, customer satisfaction, and team work to achieve long-term success.

How can TQM can be applied in S/w Development

1. Customer-centric Approach :
   - focus on customer needs and expectations during requirement gathering.

2. Process-Oriented Approach :
   - Streamline development process using models like ISO 9001 or CMM.

3. Continuous Improvement :
   - Regularly refine methods like Agile & DevOps using feedback from testing and development.

4. Employee Involvement :
   - Encourage teamwork and provide training for skill & quality improvement.

5. Quality Metrics :
   - measures defects density, test coverage, and cycle time for improvement

6. Tools and Techniques :
   - Use tools for version control, testing and statistical process monitoring.

7. Defect Prevention : prevent defects with peer reviews, code inspections, and static analysis.

8. Leadership support :
   Top management must drive and align quality goals with organizational objectives

Q3. What are the key requirements of ISO 9001 for Software development, and how do they help ensure the consistency and quality of s/w products?

Soln. Key Requirements of ISO 9001 for s/w Development

1. Quality Management System (QMS):
   - Establish and document processes, policies and objectives
   - Maintain records to demonstrate process effectiveness.

2. Leadership and Commitment:
   - Management must demonstrate commitment to quality goals.
   - Assign clear roles and responsibilities within the organization.

3. Customer focus:
   - Identify customer requirements clearly
   - Strive to enhance customer satisfaction

4. Risk-Based Thinking:
   - Identify risks and opportunities in s/w development
   - Implement actions to mitigate risks and seize opportunities

5. Resource Management:
   - ensure availability of skilled personnel, infrastructure, and tools.

6. Process Control:

- follow standardized process for s/w design, development, testing and deployment.
- ensure changes in requirements or designs are documented and controlled.

7. Continuous Improvement

- Implement corrective and preventive actions based on audits and feedback
- Focus on reducing defects and improving efficiency.

How These Requirement Ensure Consistency and Quality:

- Standardization: A documented QMS ensures processes are consistent across teams and projects.
- Customer Satisfaction: Clear focus on customer needs result in s/w that meets
- Risk Mitigation: Identifying and addressing risks reduces project delays and defects.
- Continuous monitoring: Regular evaluations highlight performance gaps, enabling improvements.
- Employee Competence: skilled personnel and training ensure high-quality outputs
- Improved Reliability: controlling processes & changes reduces errors and inconsistencies.

Q4 What is the significance of Six Sigma in the context of s/w quality assurance, and how does it relate to statistical methods for defect reduction?

Soln Six Sigma is a data driven methodology in s/w quality assurance (SQA) that minimize errors and variability in development & testing, ensuring reliable and high-quality s/w. Using the structured DMAIC approach (Define, Measure, Analyze, Improve, Control), (DMAIC) it systematically improves processes & reduce defects.

Relation to Statistical Methods for Defect Reduction

1. Defect Measurement:

Statistical tools are used to measure defects per million opportunities (DPMO), providing a quantitative way to assess s/w quality.

2. Process Capability Analysis:

Metrics like sigma level & process capability index (Cpk) determine how well processes meet quality standards.

3 Root Cause Analysis:

Techniques like Pareto analysis and fishbone diagrams identify the root causes of defects, ensuring targeted corrective actions.

4. Control charts:

Statistical process control (SPC) charts monitor s/w processes over time to detect deviations & maintain consistency.

5- Data Driven Decisions:

Six sigma uses regression analysis, hypothesis testing, and design of experiments (DOE) to optimize processes & reduce errors.

Impacts on Software Quality

By applying six sigma principles, organisations can achieve:

- Lower Defect Rates: Improved processes reduce coding and testing errors.
- Higher Customer Satisfaction: Reliable s/w meets or exceeds user expectations.
- Cost Efficiency: Fewer defects mean reduced rework and faster delivery.

Q5. What are the benefits and potential challenges of implementing a zero Defect approach in s/w development?

Soln Benefits of a Zero Defect Approach in S/w Development

1. Enhanced Software Quality:
   Focuses on minimizing defects, resulting in reliable and high-performing software

2. Increased Customer Satisfaction:
   Delivery defect-free products meets customer expectations and build trust.

3. Cost Savings:
   Reducing defects lower ~~costs~~ rework, debugging and maintenance costs.

4. Improved Productivity:
   Developers spends less time fixing issues, allowing them to focus on innovation and new features.

5. Brand Reputation:
   Consistently high-quality products enhance the organization's market reputation

Potential Challenges of a Zero Defect Approach

1. **High Initial Investment:**

   Requires significant resources for training, tools, and process optimization.

2. **Increased Development Time:**

   Aiming for zero defects may prolong development due to exhaustive testing & quality checks.

3. **Complexity:**

   Achieving zero defects in large, complex systems can be unrealistic due to unpredictable variables.

4. **Diminishing Returns:**

   Beyond a certain point, the effort to eliminate every defect may not justify the cost or time.

5. **Team Pressure:**

   The focus on perfection may create stress and affect team morale.

Q6. Explain the various components involved in System Test Design and planning.

Soln. Components:

1. Test Objectives:
   - Define the goal of system objective / testing.

2. Test Strategy:
   - A high level approach outlining the overall testing process, including the scope, objectives, test levels, and testing methods.

3. Test Plan:
   A detailed document that specifies the scope, resources, schedule, deliverables, and testing activities

4. Test scope:
   Clarifies what will be tested and what will not be tested.

5. Test scenario:
   High level description of test cases that cover the main functionalities of the system.

6. Test Cases:

Detailed description of individual tests to be executed including input data, expected outcomes.

7. Test Environment:

Define the hardware, s/w, network configuration, and other resources needed to perform the tests. It includes staging environment

8. Test Data:

The data required to execute test cases, which must be representative of real-world usage scenarios.

9. Test Tools:

Identifies the tools & frameworks needed for test execution, reporting & automation.

10. Risk Analysis:

Identifying potential risk related to the system

11. Defect Management:

Describe how defects discovered during testing will be tracked, documented, assigned, and managed, ensuring proper resolution.

12. Exit Criteria

Q7. Explain the various components involved in System Test Execution, and Acceptance Testing.

Sol^n System Test Execution

ensures entire works as a unified entity, meeting specified requirements.

1. Test Planning and strategy:

Define test objective, scope, resources, timelines and methodologies for comprehensive System validation.

2. Test Environment setup:

configure hardware, s/w and network to mimic the production environment for accurate testing.

3. Test Data Preparation:

Create & validate datasets representing real-world scenarios to ensure effective testcase

4. Test case execution:

execute detailed test cases covering functional, integration, and non-functional aspects.

Defect Management

Identify, document, prioritize, and resolve defects, ensuring a systematic approach to issue handling.

6. Integration Testing:

Validate all the system modules interact seamlessly & function as intended when combined.

7. Regression Testing:

Re-test previously validated functionality to ensure no adverse effects after updates or fixes.

8. Performance & Load Testing:

Assess system behavior under expected and stress conditions, ensuring reliability and scalability.

Q8. What are the different types of acceptance testing, and when are they performed explain with an example?

Soln

① User Acceptance Testing:

- Performed after system testing, before system goes live
- validate the system meets end-user's needs in real-world scenarios.
- example: A banking app is tested by customer to ensure features.

② Business Acceptance Testing:

- Perform After functional & system testing but before UAT
- ensure system align with business goals
- example: An e-commerce platform is validated by business stakeholders to ensure it supports inventory management, order tracking etc.

③ Operational Acceptance Testing (OAT):

- Performed before deployment to production, focusing on operational readiness
- purpose: Test backups, recovery, system monitoring.
- Example: Testing a health care system disaster recovery plan by simulating a server crash and verify data restoration

4. Contrast Acceptanie Testing:

- Performed during project closure
- purpose to validate dibiverables comply with terms agreed upon in the contract.
- Example: A government IT project is tested agaiv predefined security standards, performances

5. Compliance Acceptance Testing:

- Performed Before deployment, often in parallel with UAT
- Purpose to verify the system adheres to legal
- example: A payment processing system is tested to ensure it complies with "PCI DSS"

6. Alpha Testing:

- Performed In-house, during the development phase
- Identify Issues in functionality and usability internally.
- example: A s/w team tests a new gaming application to identify bug.

Q9. Explain the various test design factor that need to be considered when planning for system testing.

Soln When planning for system Testing factors:

1. Requirement Coverage:
   - ensure all functional & non-functional req. and traceable to test cases
   - example: if the requirement specifies payment processing, test cases should be include different payment methods, scenarios,

2. Test Data Design:
   - Create realistic and diverse datasets to cover all possible i/p's, including valid, invalid, and boundary conditions
   - example: for an online registration form, test i/p include valid email IDs, incorrect formats, and empty fields

3. Test Environment setup:
   - configure a test environment that replicate the production setup, including hardware, software, and network configurations
   - example: Testing an e-commerce site on different devices, operating systems and browsers

4. Risk Based Testing:

- Prioritize areas of the system with higher risks or criticality to the business.
- example: For a banking application prioritize testing transaction over ~~prioritize~~ less critical features like user profile updates

5. Performance and Load considerations:

- Design tests to evaluate system performance under varying loads and stress conditions.
- example: Test how an airline booking system handles peak traffic during a flash sale.

6. Integration Points:

- focus on testing the integration of different system modules and third-party APIs.
- example: In a retail system, verify the seamless integration module and the checkout module.

7. Usability and Accessibility:

- Ensure the system is user-friendly and meets accessibility standards.
- example: checks if a government portal is navigable for users with disabilities including screen reader compatibility.

8. Security Testing:

- Include test cases to identify vulnerabilities and ensure data protection
- example: Test a login system for SQL injection and brute force attacks.

9. Test Case Prioritize:

- Define the execution order of test cases based on criticality and dependencies.
- Example: Test core features like login and data input before advanced functionalities

10 Automation feasibility:

- Identify repetitive and critical test cases suitable for automation
- example: Automate regression tests for frequently changing modules in a software application.

11 Scalability and Future Needs:

- Design tests to evaluate system scalability for future growth,

example: check how a cloud-based service handles a growing number of users over time.

(10)

Describe the structure of a system test plan and its key components.

Soln It is a "system test plan" is a comprehensive document outlining the testing strategy, objectives, scopes, resources, and schedule for testing a system. It ensures systematic execution and alignment with project goals.

Key components:

1. INTRODUCTION:
   * Purpose: Define the test plan's objective and scope.
   * Example: This test plan outlines the system testing strategy for the XYZ application, ensuring it meet functional requirements and performance

2. TEST SCOPE OBJECTIVE:
   * purpose to list what the testing aims to achieve.
   * Example: Verify the accuracy of data processing, ensuring compliance with standards, and test the integration of modules.

3. TEST SCOPE:
   * Purpose to define the boundaries of testing, Specifying what is included and excluded.
   * Example: Test all core functionalities, include features in development for the next release.

4. TEST ENVIRONMENT :

* Outline hardware, software, network configurations, and setup needed for testing.

*Example: Use Windows11, Chrome browser, 8GB RAM, and a 1Gbps network.

5. TEST SCHEDULE :

* Provide a timeline for test phases including start and end dates for specific tasks.

* example: "functional testing : 1-Dec to 10-Dec, performance testing 11-Dec to 15-Dec.

6. TEST CASES and SCENARIOS:

* List test cases covering functional and non functional requirements with priority levels.

*Example : Test login functionality using valid and invalid credential, measure system response time for 1000 users

7. TEST DATA :

* Define the data sets required for testing, including edge cases and invalid inputs.

*example: Test with a dataset containing name, email & and phone numbers, including invalid format.

Roles and Responsibilities:

* Assign tasks to team members
* Example: Test execution: QA Team, Environment setup: DevOps Team, Reporting: TestLEAD.

9. Defect Management Plan:

* Define the process for logging, tracking and resolving defects
* Example: Defects will be logged in Jira, Categorized as Critical, High, Medium, or Low, and tracked daily.

10. Exit Criteria:

* Specify conditions under which testing can be concluded.
* Example: "Testing ends when all critical defects are resolved, 95% test cases pass, and stakeholder approval is obtained.

11. Risk and Mitigation

* Identify potential risk and their mitigation strategies
* Example: "Risk Delay in test environment setup. Mitigation: Prepare a backup environment plan.

12. Test Deliverables:

* List the output the testing phase

* Example: Test summary report, defect logs, and updated test cases.

13. Tools and Resources:

* Mention tools used for test management, execution, and reporting.

* Example: "Selenium for test automation, JIRA for defect tracking, and Postman for API testing

2/12/24