Department of Information Technology

CEC, Landran, Mohali.

Assignment - 1

Subject :— STQA

Subject Code - BTIT 702-18

Name - Aditya Prabhakar

Roll no - 2102560

Course - B.tech/IT

Semester — 7th

Submitted to :-

Mr. Pardeep Tiwana

Q1 :- Compare and contrast terms errors failures and faults using suitable examples.

Ans :- Errors faults, and failures are terms commonly used in the context of computers systems, engineering, and quality assurance. while they are related, they have distinct meanings and implications. Let's compare and contrast these terms using suitable examples.

1) Errors :

Errors refer to human actions or processes that result in incorrect or unintended outcomes. They are typically the result of mistakes made by individuals during design, development, or operation.

- Examples of errors :
  - Typographical errors in code or documentation.
  - Misinterpretation or requirements leading or incorrect software features.
  - Data entry mistakes resulting in incorrect information in a database.

2) Faults (Defects) :

Faults, also known as defects or bugs, are anomalies or problems in the code or systems that can

lead to incorrect behaviours or unexpected results. They are typically introduced during the development phase.

=> Example of faults :-

- A programming error that causes a software application to crash when certain inputs are provided.
- Memory leaks that gradually consume system resources, leading to performance degradation.
- Calculation errors that result in incorrect financial transaction in a banking software

=> Failures

- Failure occurs when a system or component deviates from its expected functionality or ceases to perform its intended function. Failures are observable by users or automated monitoring systems and can be the result of errors or faults.

- Examples of failures :-

A web server failing to respond to incoming requests, causing a website to be inaccessible.

- A power supply unit in a computer failing to provide the necessary voltage, causing the system to shut down unexpectedly.

-> A car engine ceasing to function due to a critical mechanical failure.

In summary, errors are human-made mistakes, faults are defects or bugs within a system, and failures are observable deviation from expected behaviour or functionality. Errors can lead to the introduction of faults, and faults can ultimately result in failure. Detection and correcting errors and faults is essential to prevents to ~~the~~ system failure and ensure the reliability and quality of products and services.

Q2: Define software quality. Discuss TQM and Six Sigma approaches for software quality.

Ans: Software Quality: Software quality refers to the degree to which a software product or system meets specified requirements, satisfies customer needs and expectations, and is free from defects or errors. It encompasses various attributes, include functionality, reliability, performance, usability, security, maintainability, and ~~same~~ scalability. Achieving high software quality is crucial to ensure the software products are dependable, effective, and meet user needs.

Total Quality Management :- TQM for software quality :-

Total Quality Management is a management approach that aims to improve the quality of products and services through continuous process improvement and customer satisfaction. When applied to software development and quality assurance. TQM focuses on several key principles:

1) Customer focus - Understanding and meeting customer's requirements is central to TQM. Software development teams should actively engage with customers to gather their feedback and ensure their needs are addressed.

2) Continuous improvement :- TQM encourages a culture of continuous improvement in all aspects of software development. Teams regularly review processes, identify areas of for enhancement, and implement changes.

3) Employee involvement :- TQM emphasizes the importance of involving all team members in quality improvement efforts. It encourages collaboration, training, and empowerment of employees to make decisions that positively impact software quality.

3) Analyze: Analyze the data collected to identify the root causes of defects or variations in software quality. Tools such as root cause analysis and statistical analysis help pinpoint the source of problems.

4) Improve: Based on the analysis, make necessary improvements to software processes to eliminate defects and improve quality. This may involve iterative experimentation and process optimization.

5) Control: Implement control measures to ensure that the improvements are sustained over time. Develop monitoring systems and standards to prevent regression.

In summary, both TQM and Six Sigma provide systematic approaches for achieving and maintaining high software quality.

4. Process Management - TQM advocates for well-defined and controlled processes. Software development processes should be standardized, documented, and optimized to reduce defects and improve efficiency.

5.) Data-Driven decision Making - TQM relies on data and metrics to assess and improve quality. Teams collect and analyze data to identify trends, defects, and opportunities for improvement.

Six Sigma for Software Quality -

1) Define - Clearly define the project's goals and objectives, along with customer requirements. This phase ensures a thorough understanding of the problem and sets measurable targets for improvement.

2) Measure - Measure current software performance using relevant metrics and data collection techniques. This step involves identifying key performance indicators (KPI's) and establishing a baseline.

Q3 :- How can you identify the causes and effects of the risks in your company?

Ans :- Here's how you can identify the causes and effects of risks:

Identifying the Causes of Risks :-

1) Brainstorming :- Gather a cross-functional team of experts and stakeholders to brainstorm potential risks. Encourage open and creative discussions to identify various risk factors.

2) Root cause Analysis :- Use techniques like the "5 Whys" methods to delve deep into the cause of risks. Ask "Why" multiple times to trace a risk back to its root cause. This helps in identifying underlying issues.

Historical Data Analysis :-

Examine past projects, incidents, or issues in your company to identify recurring patterns or common causes of problems. Historical data can reveal

trends and risk factors

4) Documentation Review - Review project documents, reports, incident logs, and other relevant records to identify and documented causes of past risks or issues.

5) Expert Interviews - Consult subjects matters experts within your organization to gain insights into potential risks and their underlying causes. Experts often have valuable knowledge about specific risks in their domain.

6) External Benchmarking :- Compare your company's risk landscape to industry benchmarks and best practices to identify potential causes of risks that might be specific to your industry.

7) Process Analysis :- Analyze your company's internal processes to identify process-related risks. Poorly designed or executed processes can be significant risk factors.

8 Environmental Scanning - Monitoring external factors such as economic conditions, regulatory changes, market trends, and geopolitical events. Changes in the external environment can lead to new risk causes.

Identifying the effect of Risks:

1) Risk Impact Assessment: Assess the potential consequences of identified risks. Consider the financial, operational, reputational, legal, and strategic impacts that a risk could have on your organization.

2) Scenario Analysis - Create scenarios that outline the effect of different risk outcomes. This can help you visualize the potential consequences and prepare mitigation strategies accordingly.

3) Historical Data Review - Examine historical data and records to understand the actual effects of past risks on your company. This can provide valuable insights into the types and magnitudes of impacts.

4) Simulation and Modeling - Use risk modeling or simulation tools to simulate the effects of various risk scenarios. This can help in quantifying potential impacts and making informed decisions.

(5) Expert Opinions - Consult experts and experienced individuals within your organization to gain insights into the potential effects of identified risks. Their expertise can provide a realistic perspective.

6) Stakeholder Feedback: Engage with stakeholders including customers, employees, and suppliers, to gather their input on the potential effects of risks. Different stakeholders may have different perspective.

7) Regulatory and Compliance Analysis:
Review regulatory requirements and compliance standards applicable to your industry. Understand how non-compliance or regulatory changes can impact your company.

Market Research:
Conduct market research to identify potential effect of market-related risks, such as changes in customer demand or competitive dynamics.

Business Impact Analysis (BIA)
Perform a BIA to assess the effects of risks on critical business functions and processes. This is particularly important for continuity and disaster recovery planning.

Q4: Examine the tester role in software development organization.

Ans: The role of a tester in a software development organization is crucial for ensuring the quality, reliability, and functionality of software products. Testers play a significant part in the software development lifecycle by identifying and reporting defects, validating that software meets requirements, and helping deliver a product that meets customer expectations. Here's an examination of the tester role in a software development organization:

1) Requirement Analysis:

Testers collaborate with business analysts and stakeholders to understand software requirements. They review requirements to identify testable elements, potential ambiguities, and areas that require clarification.

Test Planning: Testers participate in test planning activities, including creating test strategies, test plans, and test cases. They determine the scope of testing, testing objectives, and criteria of success.

3) Test Design - Testers design test cases and test scenarios based on the requirements and specifications of the software. They consider different test levels (Unit, integration, system, acceptance) and design appropriate tests for each level.

4) Test Environment setup - Testers ensures that the necessary test environment (hardware, software, data) are setup and configured correctly to mimic the production environment as closely as possible.

5) Test Execution - Testers execute test cases, record test results, and report defects. They verify that the software functions correctly and that it meets the specified requirements.

Regression Testing - Testers perform regression testing to ensure that new code changes or updates do not introduce new defects or negatively impact existing functionality.

Automated Testing - Testers may use automation tools to create and execute automated test scripts for repetitive and regression testing. This

helps improve efficiency and coverage.

8.) Defect Reporting: Testers document and report defects they discover during testing. They provide detailed information to developers, including steps to reproduce issues and their severity.

7) Collaboration - Testers work closely with developers, project managers, and other team members to resolve defects, clarify requirements, and ensure a common understanding of quality expectations.

6) Continuous Improvement - Testers continually assess and improve testing process, methodologies, & tools. They seek opportunities to enhance the efficiency and effectiveness of testing practices.

5) Test Documentation :-
Testers maintain documentation related to test cases, test plans, and test results. This documentation is important for traceability and compliance purposes

12) Validation and Verification - Testers validate that the software meets customers needs and verifies that it adhere to quality standards, design specification, and regulatory requirements.

13) User Acceptance Testing (UAT) - In some cases, testers coordinate and support UAT with end-users or clients to ensure that the software aligns with user expectations & business objectives.

14) Security Testing - Testers may specialize in security testing to identify vulnerabilities and security risks with in the software and provide recommendations for mitigation.

15) Risk Assessment - Testers assess the risks associate with different aspects to the software, helping stake holders make informed decisions about release readiness.

Q5: Construct the various black box test
Equivalence class partitioning and boundary value
analysis to test a module of payroll system

Ans :- Equivalence class partitioning and boundary value analysis are two common techniques for generating black box test cases. In the context of testing a module of a payroll system. Let's consider a simple example where you need to test a function that calculates an employee's bonus based on their years of service. Here's how you can use these techniques to create test cases:

Module: CalculateBonus(yearsOfService)

Equivalence Class Partitioning:

Equivalence class partitioning divides input data into different equivalence classes or groups. where the behaviour of the system should be the same for all members of a class. Here, we'll consider three equivalence classes:

1) Valid Years of Service (Positive Values):

- Input values representing years of service greater than zero.
- Equivalence class: [1, 2, 3 ...]

2) Invalid Years of Services (Non-positive Values):

- Input values representing year of service less than or equal to zero.
- Equivalence class: [0, -1, -2, ...]

Extreme values (Boundary Cases):

- Input values representing the boundary conditions (e.g., zero, very large values)

Equivalence class: [0, MAX_INT] (where MAX_INT represents the maximum allowable value for years of service).

Boundary Value Analysis :

Boundary value analysis focuses on testing value near the boundaries of equivalence classes as these are often where errors occurs. For example, we'll consider the following boundary values :

1) Valid Boundary Value :

- Test with the minimum valid input value.
- Test case : CalculateBonus(1)

2) Invalid Boundary Values :

- Test with the minimum & maximum invalid input values

Test cases : CalculateBonus(0), CalculateBonus(-1), CalculateBonus(MIN_INT) (~~where~~ where MIN_INT represents the minimum allowable value for years of service).

3) Extreme Boundary Value:

- Test with the maximum valid input value -
- Test cases = CalculateBonus (MAX_INT) (where MAX_INT represent the maximum allowable value for years of service)

4) Boundary value Around Zero:

- Test value near the boundary between valid and invalid inputs.

Test cases: CalculateBonus (0,1), Calculate (-0,1)