SANGAM UNIVERSITY

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

NAME: Harshvardhan Gandhare

ENROLLMENT NO: 2021BTCS008

SESSION: 2024-25 COURSE: B.Tech

BRANCH: Computer Science SEMESTER: VII

SUBJECT: Software Testing & Quality Assurance

SUBJECT CODE: PEC-CS-13

GUIDED BY: Prof. Raj Kumar Lomani

Verified / Checked

Signature

SANGAM UNIVERSITY
where Aspiration meets Opportunity

Q1 Explain role and objective of testing also verification and validation in details!

Ans

Role and Objective of Testing.

- **finding bugs:** The main role of testing is to find the mistakes or errors in the software so it works as expected.
- **Ensuring quality:** Testing makes sure the software is reliable, works well, and meets the user's expectations.
- **Reducing Risks:** Testing helps avoid issues like system crashes or security problems by identifying potential risks early on.
- **Meeting requirements:** testing checks if the software meets the initial requirements and includes all the necessary features.
- **Saving costs:** finding and fixing errors early during development is cheaper than fixing them later.

2-2-23 5 16

Objective of Testing:

- Check functionality: The goal is to find/fix bugs make sure the software works as expected in different situations
- Check Performances: Testing ensures the software performs well, even when it's under pressure (like with heavy usage)
- Check security: Testing looks for security flaws that could lead to issues like data theft.
- Check Usability: The software should be easy to use for the end users.
- Check compatibility: Testing ensures the software works properly across different devices, browsers, and systems.

Verification and Validation

Verification:

1. Verification: It checks if the software is being built correctly, following the plans and design.
2. Objective: The goal is to make sure the product is developed according to the design specifications.
3. Example: Reviewing code or designs to confirm they match the plans.

Validation:

- Validation checks if the final software meets the user's needs and works as it should in real-world situations.
- objective: The goal is to confirm the software does what the user wants.
- examples: Testing of software in a real environment to see if it meets the users requirements.

details?

Verification checks if you're building the product right, while Validation checks if you're building the right product.

Q2. Describe testing level & testing activates with help of testing life cycle.

Ans

Testing levels.

1. Unit testing:
   - objective: To test individual units or components of the software to ensure they work as expected.
   - Performed by: Typically, developers perform unit testing
   - Tool used: JUnit, NUnit, pytest.
   - Example: Testing a single function in a program to ensure it gives the correct o/p for various i/p.

2. Integration Testing:

- Objective: To test how different module or components of the software work together.
- Performed by: Developers or testers
- Types:
  - Big Bang: Testing all components at once after all modules are completed.
  - Incremental: Testing module step-by-step, one by one.
- Tool used: Junit, TestNG.
- Example Activity: Testing how a login module interacts with a database.

3. Acceptance Testing:

- To test the s/w from the user perspective to see if it meet their needs and expectation
- Performed by end users or clients.
- Types:
  - Alpha testing: Conducted by internal team before release
  - Beta testing: Conducted by actual user in a real environment after the s/w is mostly completed.

- Tool used: UAT tools or manual testing
- Example Activity: End-users testing an app to verify if it meets their business needs

4. System Testing:
   - To test the entire system as a whole to ensure it meets the specified requirements
   - Performed by "QA team"
   - Types:
     - functional testing: Verifying features as per the requirements.
     - Non-functional testing: testing performance, usability, and security.
     - Tool used: Selenium, JMeter
     - Example Activity: Testing a complete web application to check its behavior from end to end.

Testing Life Cycle and Activities

The Software Testing Life Cycle (STLC) is a structured process that defines the phases involved in testing S/W. Each phase has specific goals and deliverable.

(A) Requirement Analysis:

- objective: Understand what needs to be tested.
- Activities:
  - Analyzing the requirement from a testing perspective.
  - Identifying testable aspects (functional & non functional requirements).
- Deliverable: Requirement traceability matrix (RTM), which maps test cases to requirement.

(B) Test planning:

- Objective: Create a test plan that outlines the testing strategy.
- Activities:
  - Estimating the time, effort and resources required for testing.
  - Defining the scope of testing (what to test and what not to test).
  - selecting testing tools and assigning roles.
- Deliverable: Test plan document.

(C) Test case Development:

- Objective: Design and create test cases.
- Activities:
  - writing test cases based on requirements
  - Preparing test data.
- Deliverable: Test cases, test scripts, & test data.

(D) Test Environment Setup:

- Objective: prepare the environment where testing will be executed.
- Activities:
  - setting up hardware & software needed for testing
  - Installing required tools and configuring servers.
- Deliverable: Test environment setup document.

(E) Test Execution:

- objective: Run the tests and record the results
- Activities:
  - Executing test cases manually or using automation tools.
  - Logging defects if any test fails.
- Deliverable: Test execution report, defect logs.

(F) Test Closure:

- objective: finalize and complete the testing process.
- Activities:
  - verifying that all tests are completed.
  - ensuring all bugs are resolved or deferred
  - Analysing test coverage, results, and lessons learned.
- Deliverable: Test closure report.

Q3. What is white box testing and black box testing explain with an example in details.

Ans

WHITE BOX TESTING

- Definition: In white box testing, the tester knows the internal code and structure of the software. The focus is on checking how the system works internally, such as testing code paths, loops, and conditions.
- example: Testing function that calculates the square of a number. The tester checks how the multiplication is done in the code and ensures it handles all i/p values correctly.
- Pros: Helps find hidden bugs and improves code quality.
- Cons: Requires knowledge of the code and can be time-consuming.

BLACK BOX TESTING

- Definition: In black box testing, the tester does not know how the internal working of the software they test the functionality by providing inputs and checking outputs based on the system's requirements.

- Example: Testing a login page by entering valid and invalid username and passwords to check if the system behaves as expected.
- Technologies: Equivalence partitioning, boundary value analysis.

Q4. Explain concept of static and dynamic unit testing with examples.

Ans. STATIC UNIT TESTING: It involves reviewing code without executing it to catch syntax errors, vulnerabilities, and inefficiencies. This is done through techniques like code reviews or using static analysis tools. For example, reviewing a divide(a, b) function for issues like dividing by zero before running the code.

DYNAMIC UNIT TESTING: On the other hand, involves running the code to ensure it works correctly with various inputs. For example, testing the divide(a, b) function by executing it with different inputs to verify correct results or handle errors like division by zero.

Static testing checks the code's structure, while dynamic testing verifies its behaviour when executed. Both are crucial for quality software development.

Q5. Explain control flow graph, paths, cyclomatic complexity with examples.

Ans

Control flow Graph (CFG)

A CFG is a visual representation of all possible paths in a program. Each node represents a block of code, and the edges show how the flow of control moves between these blocks.

Key components:

- Node: Represent basic blocks of code.
- Edges: Directed connections showing the flow of control.
- Entry Block: The starting point of the program flow.
- Exit Block: the endpoint where control leaves the program.

PATHS

A path in CFG is a sequence of nodes from the start of the program to the end. Each path represents a different flow through the code.

CYCLOMATIC COMPLEXITY

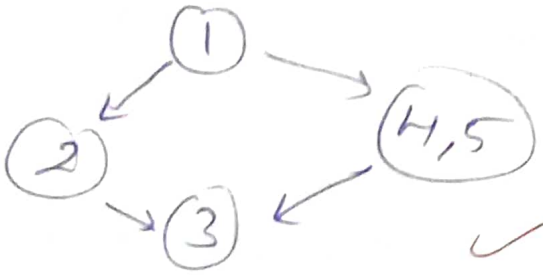
It is a way to measure how complex a program is, based on the number of independent paths through the CFG, It helps determine how many test cases are needed to cover all paths.

Example:

Consider the following pseudocode.

```
1.  if (n > 0) {
2.          print ("positive");
3.  } else {
4.          print ("Non-positive");
5.  }
6.  print ("Done");
```

CFG:



PATHS: 1 - 2 - 3

1 - 4 - 5 - 3

$V(G) = E - N + 2P$

$V(G) = 4 - 4 + 2 \times 1 = 2$

$E = 4$, $N = 4$, $P = 1$

Testcases !

Q6. Describe Data flow Anomaly, data flow terms and dynamic data flow testing.

Ans Dynamic Data flow testing:

It is a testing technique that focuses on executing the program and monitoring how data flows during execution. It helps identify data flow anomalies by observing variable definitions, uses, and their interactions at runtime.

Process

1. Test Case Design: Create test cases that cover various data flow scenarios, ensuring that definitions and uses are properly tested.
2. Execution: Run the program with designed test-cases
3. Monitoring: Track the flow of data, noting when variables are defined and used, and check for anomalies.
4. Analysis: Review the collected data to identify any issues in how data is managed.

Data flow Anomaly

It occurs when there are issues in how data is used, defined, and managed throughout a program. These anomalies can lead to unexpected behaviour, incorrect results, or security vulnerabilities.

Types of Data flow Anomalies.

1. Definition-Use Anomaly: This occurs when a variable is defined but not used before it is redefined, which may lead to confusion or unintended behaviour.

2. Use- Definition Anomaly: This happens when a variable is used before it is defined, leading to errors.

3. Redefinition Anomaly: This occurs when a variable is redefined without being used in between, which might lead to loss of the original value.

Data flow Terms:

1. Definition: this refers to the point in the code where a variable is declared or assigned a value
   - Example x = 10 is a definition of x
2. Use: This is when a variable is accessed or utilized in some operation
   - example print(x) is a use of x
3. Data flow Path: This is the route that data takes from its definition to its use. Tracking this path helps identify anomalies.
4. Data flow Analysis: This is a process of examining how data moves through a program to find anomalies and improve data management

Q7. Explain the concept of integration testing, different types of interfaces and errors.

Ans

Integration Testing.

It is a type of integration testing where individual modules or components are tested together as a group to ensure that they work correctly when combined. It verifies the interaction between integrated units to detect any interface errors, communication problems or data issues.

The primary goal of integration testing is to identify errors in the interaction between modules that may be apparent during unit testing which tests individual components in isolations.

Types of interfaces in Integration Testing.

1. Procedure Call Interface:

This occurs when one module can calls another directly, passing data as parameters or returning result. It tests whether the calling and called functions work together correctly.

eg: function A call function B and passes two parameters; the test verifies if B processes these parameters and return the expected o/p to A.

2. Shared Memory Interface:

Two or more modules share a common memory space to exchange data.

3. Message Passing Interface:

Modules communicate by sending messages or signals to each other, often used in distributed systems or microservices.

4. File System Interface:

Modules interacts by reading from or writing to files.

5. User-Interface:

The system's components interact through a graphical user interface or command-line interface.

~~TYPES OF INTEGRATION TESTING~~

TYPES OF ERRORS IN INTEGRATION TESTING

1. Interface Mismatch:

this occurs when one module expects a specific input format or type, but another module provides incorrect or mismatched data,

2. Communication failure:

error occurs when modules fails to communicate correctly due to protocol or message format issue.

3. Data flow Issues:

Errors occur when data is not properly passed or shared between modules, such as missing or corrupted data.

4. Incorrect timing or Synchronization:

This happens when modules that need to interact in real-time or in a certain sequence fail to do so.

5. Dependency Errors:

These arises when one module is depends on another for certain resources or services that are not provided correctly.

Q8. Explain function testing, boundary value analysis, decision tables.

FUNCTIONAL TESTING

It is a type of software testing that focuses on verifying that the software behaves according to its specified functionality. It checks if the system meets all functional requirements by testing individual functions or features to ensure they work as expected.

Example

In a login feature, functional testing would verify:

- correct username or password allows the user to login
- Incorrect username or password prevent login.
- The system behaves as expected when the "forgot password" link is clicked.

BOUNDARY VALUE ANALYSIS (BVA)

It is a testing technique used to identify errors at the boundaries of input ranges, where errors are most likely to occur. It focuses on testing the edges or boundaries of input domains rather than the middle value.

Example: For a system that accepts input between 1 and 100, the boundary values would be:

- Lower boundary: 0, 1, 2 (just below, at, & just above the lower limit).
- Upper Boundary: 99, 100, 101 (just below, at, & just above the upper limit).

⇒ In this case:

- 1 & 100 are valid boundary values
- 0 & 101 are invalid values, & the system should be to handle them properly.

Decision Table

Decision Table is a tool used for representing and analysing complex decision-making logic, particularly in scenarios with multiple conditions and actions. It helps systematically test all combinations & the resulting actions.

Structure: Decision table consists of:

- Conditions: Define factors or inputs that can influence the outcome
- Actions: The possible outcomes or actions based on the conditions
- Rules: Specific combinations of conditions that lead to particular actions.

Examples: Consider an online shopping system with the following conditions:

1. If the user logged in?
2. Does the user have sufficient balance.

The actions are:

1. Proceed with the purchase 2. Show error message

Condition1: Logged In	Condition2: Sufficient Balance	Action
Yes	Yes	Proceed with purchase
Yes	No	Show error msg.
No	Yes	Show error msg.
No	No	Show error msg

/ Q9

Test Plan

"Online examination System"

1. Introduction

Software: *Online Examination System*
Prepared By: *Harshvardhan Gandharv*
Date: *October 15, 2024* Version: *1.0*
This document outlines the test plan for the "*Online Examination System*", detailing the testing strategy, objectives, resources, schedule, and specific test cases to ensure the system works as intended.

2. Objectives

- Ensure that all modules function correctly and meet requirements.
- Validate the reliability and security of the system.
- Test the usability and user interface (UI).
- Ensure performance under varying loads.

3. Scope

- Functional testing (login, examination management, question bank, results).
- Security testing (data protection, user authentication).
- Performance testing (handling simultaneous users).
- Usability testing (ease of use, accessibility).

4. Resources

- Testers: 3 members.
- Testing environment: Windows/Linux with Chrome, Firefox, and Edge browsers.
- Tools: Selenium (for automation), JMeter (for load testing), Manual testing tools.

5. Test Environment

- Operating System: Windows, Linux, macOS.
- Browsers: Chrome, Firefox, Edge.
- Database: MySQL or PostgreSQL.

6. Features to be Tested

- **User Management**: Registration, Login, Password Recovery.
- **Examination Creation**: Admin functionalities to create and manage exams.
- **Taking Exams**: Question navigation, timer functionality, submission.
- **Result Generation**: Immediate results for objective questions, delayed results for subjective ones.
- **Security**: Data encryption, secure login, and prevention of cheating.

7. Test Cases

Module: User Management

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_01	User Registration	Browser is open	1. Go to the registration page. 2. Fill out the form. 3. Submit.	A new user account was created successfully and redirected to login.
TC_02	Login with valid credentials	User is registered	1. Navigate to the login page. 2. Enter valid credentials. 3. Click Login.	After a successful login, the user dashboard is displayed.
TC_03	Login with invalid credentials	User is registered	1. Enter invalid credentials. 2. Click Login.	Error message: "Invalid username or password."
TC_04	Password Recovery	The user has a registered email	1. Click "Forgot Password". 2. Enter registered email. 3. Submit.	An email with a reset link is sent to the user.

Module: Examination Management

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_05	Admin creates an exam	Admin is logged in	1. Go to the "Create Exam" page. 2. Fill out exam details. 3. Save.	Exam created and listed under "Upcoming Exams".
TC_06	Admin edits an exam	Exam exists	1. Select the exam to edit. 2. Make changes. 3. Save.	Changes saved successfully.
TC_07	Admin deletes an exam	Exam exists	1. Select an exam. 2. Click "Delete".	The exam is removed from the list.

Module: Taking Exams

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_08	The user starts an exam	The user is logged in and enrolled in the exam	1. Navigate to "My Exams". 2. Select an exam. 3. Start the exam.	An exam page with questions and a timer is displayed.
TC_09	The user answers all questions	The user is taking the exam	1. Answer all questions. 2. Click "Submit".	Answers were submitted successfully, and a confirmation message was shown.
TC_10	The user navigates between questions	The user is taking the exam	1. Start the exam. 2. Click "Next" and "Previous".	Question navigation works correctly.
TC_11	Timer expiration	The user is taking the exam	1. Start the exam. 2. Wait until the timer expires.	The exam auto-submits, and the user is redirected to the result page.

Module: Result Generation

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_12	Instant result for objective exam	The user completed the objective exam	1. Submit exam. 2. View results.	Users can see the score immediately.
TC_13	Result of subjective exam	The user completed the subjective exam	1. Submit exam. 2. Wait for evaluation.	Results published after admin evaluation.

Module: Security Testing

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_14	SQL Injection Test	The login form is available	1. Enter the SQL injection attack string in the input. 2. Submit.	The system rejects input and displays an error message.
TC_15	Secure data transmission	User is registering	1. Monitor network traffic during registration.	User data is transmitted securely over HTTPS.
TC_16	Cheating prevention	The user is taking an exam	1. Open another tab or attempt to leave the page during the exam.	Exam auto-submits or the user is warned about leaving the exam window.

8. Performance Testing

Test Case ID	Description	Preconditions	Steps	Expected Result
TC_17	Load testing	The system is up and running	1. Simulate 100 to 500 users taking an exam simultaneously.	The system handles the load without crashing, response times are acceptable.

9. Schedule

Testing will occur over 4 weeks:

- **Week 1**: Set up the test environment, and write test cases.
- **Week 2-3**: Execute functional and security test cases.
- **Week 4**: Perform performance and usability testing, and document results.

10. Risks

- Lack of resources for large-scale load testing.
- Unanticipated security vulnerabilities.

11. Exit Criteria

- All test cases executed with no major open defects.
- Functional, performance, and security testing are successful.

16/10/24

10. What is Software Quality? Explain quality assurance, quality control.

SOFTWARE QUALITY

It is refers to the degree to which a software product meets its requirements and satisfies the needs and expectations of its users. High-quality software is reliable, efficient, secure, maintainable, and meets all functional and non-functional requirements.

Characteristics:

- functionality: the S/w works as intended
- Reliability: It runs without errors
- Usability: It's user-friendly and easy to navigate.
- Efficiency: It uses resources like memory & processing power well.
- Maintainability: It's easy to update and fix.
- Portability: It can be easily moved to different environment.

QUALITY ASSURANCE

It is a process-focused approach aimed at ensuring that the processes used to develop S/w are effective and meet predefined quality standards. QA involves setting up proper methods, procedures, & guidelines for preventing defects in the S/w development process.

Key Activities:

- Establishing standards and process for s/w development
- Conducting process audits and reviews
- Providing training and resources to ensure processes are followed.
- Improving development methodologies over time.

Quality Control (QC)

It is a product-focused approach that involves identifying and fixing defects in the software product. It is more about testing and inspection, focusing on finding bugs and errors in the final product to ensure it meets the specified quality criteria.

Key Activities:

- executing test cases to verify s/w functionality
- conducting inspections and reviews of the s/w code.
- Reporting and tracking defects found during testing.
- Ensuring that defects are corrected before release.

16/10/24