DECCAN EDUCATION SOCIETY'S

KIRTI M. DOONGURSEE COLLEGE (AUTONOMOUS)

DADAR WEST, MUMBAI-400028.

ASSIGNMENT

NAME:- MAYURESH MANGESH MANKAR

ROLL NO.:- 64 **CLASS:-** TYCS **DIV:-** A

SUBJECT:- Software Testing & Quality Assurance

PAPER NO. _____ **SEM.** V **Year:-** 2025-2026

Date:-_____ Student's Signature_____	Mark Obtained ___ / 20 Name of Examiner_____ Signature_____

Q.1 Write a test case to verify the login functionality of a banking app.

→ Software testing plays a very important role in ensuring the reliability of banking applications, as they deal with highly sensitive financial data. One of the most essential features in such apps is the login functionality if login fails or is insecure, it may allow unauthorized access, leading to financial loss or security breaches. Hence, writing a proper test case for login is a must.

Test Case for login functionality

- Test case ID: TC_login_01
- Test Objective: To verify that the login page of the banking app allows valid users to access their account and prevents invalid users.
- Pre-conditions:
  1. The banking application must be installed or accessible through a browser.
  2. The user must have a registered username and password.

Test Steps:

1. Launch the banking application.
2. Navigate to the login page.
3. Enter valid credentials (username and password).
4. Click on the "Login" button.
5. Observe the response of the system.

Test Data:

- Username : user123
- Password : Pass@123

Expected Result:

- If the entered credentials are correct, the user should be redirected to their dashboard.
- If the credentials are incorrect, the application should display a message like "Invaid username or Password".

Conclusion:

This test case covers both positive and negative scenarios. It ensures that the login features in both functional and secure. Without testing this properly the app cannot be trusted by user.

Q2. Apply boundary value analysis to test a form accepting in which age between 18 and 60.

→ Boundary Value Analysis (BVA) is a black-box testing technique in which test cases are created around the extreme values of input domains. Many errors in progress occur at the boundaries rather than the center of input values, so BVA is one of the most widely used techniques.

Scenario: The form accepts age values between 18 and 60

- Valid Range: $18 \leq Age \leq 60$
- Boundary values: 17, 18, 19, 59, 60, 61

Test Cases:

1. Age = 17 → Invalid (Below minimum boundary).
2. Age = 18 → Valid (Minimum acceptable value).
3. Age = 19 → valid (Just above minimum boundary)
4. Age = 59 → valid (Just below maximum boundary).
5. Age = 60 → valid (Maximum acceptable value).
6. Age = 61 → Invalid (Above maximum boundary).

Analysis:

- If the form accepts 17, it means the validation is wrong.
- If the form rejects 18, the lower limit is not implemented proper
- If the form rejects 60, the upper limits is faulty.

Conclusion: By applying BVA, we test at the most error-prone edges of input. This ensures that the system correctly validates user's ages within the specified range.

Q3 Demonstrate how equivalence partitioning can be used to test a mobile number input field.

→ Equivalence Partitioning (EP) is a black-box method in which input data is divided into different classes, (partitions). Each class represents a set of valid or Invalid inputs. From each class, one value is selected for testing, reducing the total number of test cases while still ensuring effective coverage.

Scenario: The mobile number field must accept exactly 10 digits.

Equivalence classes:

1. Valid class: 10-digits numbers (eg: 9876543210).
2. Invalid classes: Less than 10 digits (eg: 98765).
   More than 10 digits (eg: 98765432109 8)
   Non-numeric inputs (eg: 98AB@3210).

Test cases:

1. Input = 9876543210 → valid (Accepted).
2. Input = 98765 → Invalid (Rejected).
3. Input = 98765432109 8 → Invalid (Rejected).
4. Input = 98AB@3210 → Invalid (Rejected).

Conclusion:

By applying equivalence partitioning, we can ensure that the mobile number field is robust and only allows valid numbers while rejecting invalid formats.

Q4. Compare black-box and white-box testing based on their strengths and limitations.

→ Software testing can broadly be divided into two approaches: Black-Box Testing and White-Box Testing. Both are important, but they differ in perspective, coverage, and application.

Black-Box Testing

- Definition: Testing without any knowledge of the internal code. The tester only provides inputs and checks outputs.
- Strengths:
  - Easy to perform without coding knowledge
  - Focuses on user requirements and functionality
  - Helps detect missing or incorrect features.
- Limitations:
  - Cannot ensure code coverage.
  - Internal error may remain undetected.
  - May required a large numbers of test cases to cover all input possibilities.

White-Box Testing

- Definitions: Testing with complete knowledge of internal code, structure, and logic. The tester examines loops, conditions, and paths in the program.

the

- Strengths:
  - Ensures maximum code coverage
  - Detects hidden logic errors.
  - Improves optimization of code.
- Limitations:
  - Requires programming expertise.
  - Time-consuming and expensive.
  - Cannot test missing Functionalities (science it focuses on existing code).
- Summary:
  - Black-Box = "What the system does" (Functionality)
  - White-Box = "How the system does it (internal logic)
- Conclusion:

Both techniques complement each other. Black-box ensures user requirements are met, while white-box ensures internal code is correct. Together, they provide complete testing coverage.

95 Distinguish between Functional and non-functional testing and Non-Functional Testing. Both are necessary to ensure the quality of testing using a practical exampl

→ Testing can be broadly divided into Functional Testing and Non-Functional Testing. Both are necessary to ensure the quality of software but focus on different aspects.

Functional Testing

- Definition: Verifies whether the system performs the required business functions correctly.
- Objective: Ensure the software behaves as expected.
- Examples of Techniques: Unit testing, integration testing, system testing, user acceptance testing.
- Practical Example: In a banking app, testing whether the "Fund Transfer" feature correctly transfers money from one account to another.

Non-Functional Testing

- Definition: Verifies the non-functional attributes of a system such as performance, security, reliability, usability, and scalability.

- Objective: Ensure the system works well under various conditions.

- Examples of Techniques: Load testing, stree testing, performance testing, security testing.

- Practical Example: In the same banking app, testing whether the system can handle 10,000 users logging in simultaneously without crashing or slowing down.

- Conclusion:

Functional testing ensure that the application is working correctly from a feature perspective, while non-function testing ensures that the system can handle real-world performance, reliability, and security needs. Both are equally important for delivering high-quality software.