Department of Computer Engineering

Academic Term: First Term 2023-24

Class: T.E /Computer Sem – V / Software Engineering

Practical No:	**8**
Title:	**Design test cases for performing black box testing**
Date of Performance:	**04/10/2023**
Roll No:	**9614**
Team Members:	**Mudabbir(9589),Muhammad(9588),Nathan(9597)**

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Total Score
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer Engineering

Academic Term: First Term 2022-23

Class: T.E /Computer Sem – V / Software Engineering

Signature of the Teacher:

Lab Experiment 08

Experiment Name: Designing Test Cases for Performing Black Box Testing in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of Black Box Testing, a testing technique that focuses on the functional aspects of a software system without examining its internal code. Students will gain practical experience in designing test cases for Black Box Testing to ensure the software meets specified requirements and functions correctly.

Introduction: Black Box Testing is a critical software testing approach that verifies the functionality of a system from an external perspective, without knowledge of its internal structure. It is based on the software's specifications and requirements, making it an essential part of software quality assurance.

Lab Experiment Overview:

1. Introduction to Black Box Testing: The lab session begins with an introduction to Black Box Testing, explaining its purpose, advantages, and the types of tests performed, such as equivalence partitioning, boundary value analysis, decision table testing, and state transition testing.
2. Defining the Sample Project: Students are provided with a sample software project along with its functional requirements, use cases, and specifications.
3. Identifying Test Scenarios: Students analyze the sample project and identify test scenarios based on its requirements and use cases. They determine the input values, expected outputs, and test conditions for each scenario.
4. Equivalence Partitioning: Students apply Equivalence Partitioning to divide the input values into groups that are likely to produce similar results. They design test cases based on each equivalence class.
5. Boundary Value Analysis: Students perform Boundary Value Analysis to determine test cases that focus on the boundaries of input ranges. They identify test cases near the minimum and maximum values of each equivalence class.
6. Decision Table Testing: Students use Decision Table Testing to handle complex logical conditions in the software's requirements. They construct decision tables and derive test cases from different combinations of conditions.
7. State Transition Testing: If applicable, students apply State Transition Testing to validate the software's behavior as it moves through various states. They design test cases to cover state transitions.
8. Test Case Documentation: Students document the designed test cases, including the test scenario, input values, expected outputs, and any preconditions or postconditions.
9. Test Execution: In a simulated test environment, students execute the designed test cases and record the results.
10. Conclusion and Reflection: Students discuss the importance of Black Box Testing in software quality assurance and reflect on their experience in designing

test cases for Black Box Testing.

Learning Outcomes: By the end of this lab experiment, students are expected to:

· Understand the concept and significance of Black Box Testing in software testing. · Gain practical experience in designing test cases for Black Box Testing based on functional requirements.
· Learn to apply techniques such as Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, and State Transition Testing in test case design.
· Develop documentation skills for recording and organizing test cases effectively. · Appreciate the role of Black Box Testing in identifying defects and ensuring software functionality.

Pre-Lab Preparations: Before the lab session, students should familiarize themselves with Black Box Testing concepts, Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, and State Transition Testing techniques.

BLACKBOX TESTING

Test Cases for Farmer Helper Website:
1. User Registration:

Test Case 1: Valid Registration

Input: User provides valid information (name, email, password).
Expected Output: User is successfully registered, and the system redirects to the dashboard.
Notes: This test case verifies that the user registration process functions correctly.

Test Case 2: Invalid Email Format

Input: User provides an email in an invalid format.
Expected Output: Registration fails, and an error message prompts the user to enter a valid email.
Notes: Ensures that the system correctly handles invalid email formats.

2. Farm Management:

Test Case 3: Adding a New Farm

Input: User adds a new farm with valid details.
Expected Output: The farm is successfully added to the user's profile.
Notes: Verifies that users can add farms to their profiles.

Test Case 4: Removing a Farm

Input: User removes an existing farm.
Expected Output: The farm is deleted from the user's profile.
Notes: Ensures that users can successfully remove farms from their profiles.

3. Crop Recommendation:

Test Case 5: Recommending Crops Based on Location

Input: User enters a farm location, and the system recommends suitable crops.
Expected Output: The system provides accurate crop recommendations based on the entered location.
Notes: Validates the accuracy of the crop recommendation algorithm.

Test Case 6: Updating Crop Preferences

Input: User updates preferences for a specific crop.
Expected Output: The system considers updated preferences in subsequent crop recommendations.
Notes: Checks if the system correctly incorporates user preferences.

4. Weather Integration:

Test Case 7: Fetching Current Weather

Input: User requests the current weather for a farm location.
Expected Output: The system displays the current weather details accurately.
Notes: Verifies the accuracy of the weather information retrieved from the external API.

Test Case 8: Forecast Accuracy

Input: User checks the weather forecast for the next week.
Expected Output: The system provides an accurate weather forecast for the specified period.
Notes: Ensures that the weather forecast functionality is reliable.

5. Decision Table Testing (Combining Conditions):

Test Case 9: Crop Recommendation with Weather Conditions
Input: User requests crop recommendations considering both location and weather conditions.
Expected Output: The system provides suitable crop recommendations considering both factors.
Notes: Tests the system's ability to integrate multiple conditions for crop recommendations.

6. State Transition Testing:

Test Case 10: Crop Growth Stage Transition
Input: User updates the growth stage of a crop in the system.
Expected Output: The system accurately reflects the transition to the updated growth stage.
Notes: Verifies that the system handles transitions in crop growth stages correctly.

7. Security and Permissions:

Test Case 11: Unauthorized Access Attempt

Input: A user attempts to access another user's farm without proper permissions.
Expected Output: The system denies access and displays an appropriate error message.
Notes: Ensures that the system correctly enforces security measures.

Test Case 12: Password Change

Input: User changes their password.
Expected Output: The password is successfully updated, and the user can log in with the new password.
Notes: Checks that the password change functionality works as intended.

Metrics and Calculations:
Success Rate of Registrations:

Metric: Percentage of successful user registrations.
Calculation:
Success Rate =(Number of Successful Registrations/Total Registration Attempts)×100

Purpose: Evaluate the effectiveness of the registration process and identifies potential issues if success rates are low.
Accuracy of Crop Recommendations:

Metric: Percentage of accurate crop recommendations.
Calculation:
Accuracy=(
Total Number of Recommendations/
Number of Accurate Recommendations)×100.
Purpose: Assesses the reliability of the crop recommendation algorithm, ensuring that users receive relevant and precise suggestions.
Farm Addition and Removal Success Rates:

Metric: Percentage of successful additions and removals of farms.
Calculation:
Success Rate =(Total Attempts/Number of Successful Additions or Removals)×100.
Purpose: Measures the usability and effectiveness of the farm management features, helping identify any challenges in adding or removing farms.

Weather Information Accuracy:

Metric: Percentage of accurate weather information.
Calculation:
Accuracy =(Total Weather Requests/Number of Accurate Weather Reports)×100.
Purpose: Ensures that the system provides reliable and up-to-date weather information, crucial for agricultural planning.
Password Change Success Rate:

Metric: Percentage of successful password changes.
Calculation:
Success Rate =(Total Attempts/Number of Successful Password Changes)×100.
Purpose: Assesses the functionality and user-friendliness of the password change feature, ensuring that users can securely manage their accounts.
Security Measures Effectiveness:

Metric: Number of unauthorized access attempts blocked.
Purpose: Measures the effectiveness of security measures in preventing unauthorized access. Ideally, this number should be minimal or zero.
Overall System Reliability:

Metric: System uptime and responsiveness.
Purpose: Tracks the overall reliability and availability of the Farmer Helper Website. Downtime or sluggish responsiveness can impact user experience and satisfaction.
Note:
Regularly monitoring these metrics over multiple test iterations and in real-world usage scenarios provides valuable insights into the performance, usability, and reliability of the Farmer Helper Website.
Any deviations from expected metrics should prompt further investigation and potential adjustments to improve the system's functionality and user experience.
These metrics are essential for ensuring that the Farmer Helper Website meets user expectations and delivers a high-quality experience.

Conclusion: The lab experiment on designing test cases for Black Box Testing provides students with essential skills in verifying software functionality from an external perspective. By applying various Black Box Testing techniques, students ensure comprehensive test coverage and identify potential defects in the software. Their experience in designing and executing test cases enhances their ability to validate software behavior and fulfill functional requirements. The lab experiment encourages students to incorporate Black Box Testing into their software testing strategies, promoting robust and high-quality software development. Emphasizing test case design in Black Box Testing empowers students to contribute to software quality assurance and deliver reliable and customer-oriented software solutions.