Department of Computer Engineering

Academic Term: First Term 2

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

Practical No:	8
Title:	Black Box testing
Date of Performance:	03 /10 /2023
Roll No:	9605
Team Members:	Rahul Gandla, Mohtashim Ali, Aditya Dhikale ,Siddhant Murade

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

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.

Materials and Resources:

□ Project brief and details for the sample software project

□ Whiteboard or projector for explaining Black Box Testing techniques

□ Test case templates for documentation

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. The experience in designing and executing test cases enhances theirability 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.

Test case1:

Test Case ID	MA_007	Test Case Description	Test the Login Functionality Shopping Website		
Created By	Mohtashim Ali	Reviewed By	Aditya Dhikale	Version	2.1

QA Tester's Log	Review comments from Bill incorprate in version 2.1

Tester's Name	Siddhant Murade	Date Tested	September 26, 2023	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	Stable Internet Connectivity
3	
4	

S #	Test Data
1	Userid = Mohtashim_1910
2	Pass = 12345678
3	
4	

Test Scenario	Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to https://adityadhikale.github.io/	Site should open	As Expected	Pass
2	Enter Userid & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Cutomer is logged in	As Expected	Pass

Test case 2:

Test Case ID	AD_001	Test Case Description	Test the Login Functionality Shopping Website		
Created By	Aditya Dhikale	Reviewed By	Siddhant Murade	Version	2.1

QA Tester's Log	Review comments from Bill incorprate in version 2.1

Tester's Name	Mohtashim Ali	Date Tested	September 26, 2023	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	Stable Internet Connectivity
3	
4	

S #	Test Data
1	Userid =Aditya_2003
2	Pass = 12345678
3	product_id=sneakers_007
4	

Test Scenario	Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to https://adityadhikale.github.io/	Site should open	As Expected	Pass
2	Enter Userid & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Cutomer is logged in	As Expected	Pass

Test case 3:

Test Case ID	SM_002	Test Case Description	Test the Login Functionality Shopping Website		
Created By	Siddhant Murad	Reviewed By	Mohtashim Ali	Version	2.1
QA Tester's Log	Review comments from Bill incorprate in version 2.1				
Tester's Name	Aditya Dhikale	Date Tested	September 26, 2023	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	Stable Internet Connectivity
3	
4	

S #	Test Data
1	Userid =Sid_2003
2	Pass = 87654321
3	product_id= sneakers_007
4	

Test Scenario: Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to https://adityadhikale.github.io/	Site should open	As Expected	Pass
2	Enter Userid & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Cutomer is logged in	As Expected	Pass

Postlabs:

Create a set of black box test cases based on a given set of functional requirements, ensuring adequate coverage of different scenarios and boundary conditions.

Ans: Creating black box test cases involves testing a system without any knowledge of its internal workings, focusing solely on the functional requirements. To ensure adequate coverage of different scenarios and boundary conditions, you can follow these general steps:

1. Understand the Functional Requirements:

- First, thoroughly understand the functional requirements provided.

2. Identify Test Scenarios:

- Identify different test scenarios based on the functional requirements. Consider the main functionalities and features that the system must provide.

3. Boundary Value Analysis:

- Identify boundary conditions for various input parameters or values. These are often the extreme values that could cause issues if not handled correctly.

4. Equivalence Partitioning:

- Divide input data into equivalence classes. Test cases should include representatives from each class, ensuring that you test a range of possible inputs.

5. Positive and Negative Testing:

- Create test cases that validate the expected behavior (positive testing) and test cases that provoke errors or exceptions (negative testing).

6. Error Handling:

- Focus on how the system handles errors and exceptions. Create test cases to check if error messages are displayed correctly and if the system recovers gracefully from errors.

7. Combination Testing:

- Test the system with combinations of different inputs or conditions, especially if the system's behavior depends on multiple variables.

8. User Role-Based Testing:

- If the system has different user roles (e.g., admin, regular user, guest), create test cases that cover the functionality specific to each role.

9. Data Validation and Security Testing:

- Check how the system handles data validation, including valid and invalid data. Also, test for security vulnerabilities if relevant.

10. Performance Testing:

- If performance is a concern, create test cases that simulate a high load on the system to ensure it can handle it.

11. Usability Testing:

- Test the user interface for ease of use and adherence to user experience (UX) design guidelines.

12. Compatibility Testing:

- Ensure the system works correctly on different browsers, operating systems, and devices.

13. Regression Testing:

- Verify that new updates or changes to the system do not break existing functionality.

14. Localization and Internationalization Testing:

- If applicable, test how the system performs with different languages, date formats, and cultural differences.

15. Integration Testing:

- Test how the system interacts with other systems, if relevant.

16. Interoperability Testing:

- If the system needs to work with third-party software or hardware, verify compatibility.

17. Accessibility Testing:

- Check if the system is accessible to users with disabilities, such as screen readers or keyboard navigation.

18. Data Migration and Recovery Testing:

- If data migration is part of the requirements, ensure that data is migrated correctly, and test data recovery procedures.

19. Scalability Testing:

- Assess how well the system scales with increased user loads or data.

20. User Acceptance Testing (UAT):

- Finally, involve end-users or stakeholders to perform UAT to ensure the system meets their expectations. Remember to document these test cases clearly, including input data, expected results, and any specific conditions. This documentation is essential for reporting and tracking issues. Test cases should cover a combination of these strategies to provide comprehensive coverage and ensure the software functions correctly in different scenarios and boundary conditions.

Evaluate the effectiveness of black box testing in uncovering defects and validating the software's functionality, comparing it with other testing techniques.

Ans: Black box testing is a widely used software testing technique that focuses on evaluating the functionality of a software application without knowledge of its internal code structure. It is performed by treating the software as a "black box," where testers only interact with the inputs and outputs, without considering the internal code logic. The effectiveness of black box testing can be evaluated by comparing it to other testing techniques, such as white box testing and gray box testing.

1. Effectiveness of Black Box Testing:

a. Objective Validation: Black box testing is highly effective in objectively validating the software's functionality. Testers do not need to know the internal codebase, which makes it suitable for independent validation. This method ensures that the software works as expected from the user's perspective.

b. User-Centric Approach: Black box testing is valuable for focusing on the end user's point of view. It helps identify issues related to usability, requirements fulfillment, and overall user experience.

c. Diverse Input Scenarios: Testers can create a wide range of test cases to simulate various user inputs and real-world scenarios. This diversity can uncover defects that might not be apparent through other testing methods.

d. Integration Testing: Black box testing is effective for integration testing because it can identify issues in data flow and interactions between different system components.

e. Independent Testing: Since it doesn't require knowledge of the internal code, black box testing can be performed by testers who are not developers, making it independent and reducing potential bias.

2. Comparison with Other Testing Techniques:

a. White Box Testing:

- White box testing is effective in uncovering defects related to the internal code structure.

- It is better suited for code coverage analysis and logic-based testing.

- White box testing can identify issues like code vulnerabilities and security flaws that black box testing might miss.

b. Gray Box Testing:

- Gray box testing combines elements of both black box and white box testing.

- It is effective for uncovering defects related to the system's architecture, data flow, and integration.

- It provides a balance between user-centric testing and code-based testing.

In summary, black box testing is highly effective for uncovering defects and validating the software's functionality from a user's perspective. Its focus on the software's external behavior makes it a valuable testing technique for requirements validation, usability testing, and integration testing. However, it should be used in conjunction with other testing methods like white box and gray box testing to ensure comprehensive testing and defect identification, especially for issues related to internal code structures and security vulnerabilities. The choice of testing technique should depend on the specific goals and requirements of the testing process.

Assess the challenges and limitations of black box testing in ensuring complete test coverage and discuss strategies to overcome them.

Ans: Black box testing is a valuable testing technique, but it does have its challenges and limitations when it comes to ensuring complete test coverage. Test coverage refers to the extent to which a test suite examines the functionalities and potential problem areas of a software application. Here are some challenges and strategies to overcome them:

Challenges of Black Box Testing for Complete Test Coverage:

1. Limited Knowledge: Testers have no access to the internal code, making it challenging to design test cases that address specific code paths and logic.

2. Inadequate Edge Cases: Black box testing may miss edge cases, exceptional conditions, and boundary values that could lead to defects.

3. Combinatorial Explosion: As the system's functionality grows, creating test cases to cover all possible input combinations becomes increasingly challenging.

4. Incomplete Requirements: If the requirements are incomplete or vague, it can be difficult to create comprehensive test cases.

Strategies to Overcome these Challenges:

1. Use Requirements Analysis: Work closely with stakeholders to ensure that the requirements are well-defined, complete, and unambiguous. Thoroughly understanding the requirements can help in designing test cases that align with the intended functionality.

2. Equivalence Partitioning: Apply equivalence partitioning to group inputs into categories that should produce similar results. This can help in reducing the number of test cases while still providing good coverage.

3. Boundary Value Analysis: Pay special attention to boundary values and test conditions near the boundaries, as defects often occur in these areas. This can improve coverage without a large number of test cases.

4. Adopt Risk-Based Testing: Focus on areas of the software that are most critical or carry the highest risk. This approach ensures that limited testing resources are directed toward the most crucial parts of the application.

5. Pair Black Box with White Box Testing: Combining black box testing with white box testing can help achieve more comprehensive coverage. White box testing can target specific code paths and logic that black box testing might miss.

6. Exploratory Testing: Allow experienced testers to explore the application and uncover defects organically. Exploratory testing can identify issues that scripted test cases might overlook.

7. Use Test Automation: Automated black box testing can execute a large number of test cases efficiently, helping to cover a wide range of input scenarios. Automation can also facilitate regression testing to ensure coverage over multiple software versions.

8. Leverage Model-Based Testing: Model-based testing involves creating a model of the system and generating test cases from the model. This can help in systematically covering various aspects of the software.

9. Continuous Improvement: Regularly review and update your test cases and strategies to adapt to changes in the software and requirements. This ensures that testing remains effective as the software evolves.

10. Test Metrics and Coverage Analysis: Implement test coverage metrics and tools to measure the effectiveness of your testing efforts. This will help identify gaps in coverage and areas that require additional testing. In conclusion, while black box testing has its challenges when it comes to ensuring complete test coverage, these challenges can be mitigated with careful planning, creative test case design, and the integration of other testing techniques. Testers should continually strive to improve their testing processes to provide more thorough coverage, but it's essential to balance coverage goals with resource constraints and project timelines.