

Department of Computer Engineering

Academic Term: First Term 2023-24

Class: 'I'.E /Computer Sem — Y / Software
Engineering

Practical No:	8
Title:	Software Requirement Specification
Date of Performance:	
Roll No:	9634
Team Members:	Rianna,Aditya,Alex,Chris

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:

A	B	C	D	E	F	G	H	I	J	K
Test Case ID	SP_003		Test Case Description	Test the cart functionality in shopping site						
Created By			Reviewed By	Prashant			Version	1.0		
QA Tester's Log	review from the website with version in 1.0									
Tester's Name	mark		Date Tested	September 26, 2023		Test Case (Pass/Fail/Not Executed)	Pass			
S #	Prerequisites:			S #	Test Data Requirement					
1	Access to Chrome Browser			1	Email id=crce.9641.ce@gmail.com					
2	stable internet connectivity			2	password=PoojaBewafaHai					
3	user must be logged in to the website			3	product id=boat_122					
4				4						
Test Conditions	verifying the search functionality by entering a product									
Step #	Step Details		Expected Results	Actual Results			Pass / Fail / Not executed / Suspended			
1	Navigate to https://shopeasy.com		Site should open	As Expected			Pass			
2	Enter Userid & Password and sign in		user should enter the credentials	As Expected			Pass			
3	select the products		user should select the products as per the requirements	As Expected			Pass			
4	Go to cart		User should be able to see there products in cart	As Expected			Pass			

Project Name:	Hospido
Module Name:	app
Reference Document:	If any
Created by:	Aditya
Date of creation:	26-09-2022
Date of review:	28-04-2023

Post lab

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.