RÉPUBLIQUE DU CAMEROUN
Paix – Travail – Patrie
****
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
****
UNIVERSITE DE YAOUNDE 1
****
FACULTE DES SCIENCES
****
DEPARTEMENT D'INFORMATIQUE
****

Faculté des Sciences

REPUBLIC OF CAMEROON
Peace - Work – Fatherland
****
MINISTRY OF HIGHER EDUCATION
****
UNIVERSITY OF YAOUNDE 1
****
FACULTE OF SCIENCE
****
COMPUTER SCIENCE DEPARTMENT
****

INF352: BLACK BOX TESTING TECHNIQUES

GROUPE 3

<u>Liste des membres :</u>

1- SOKOUDJOU CHENDJOU CHRISTIAN	21T2396
2- TSAFACK BRUNEL WEELFRED	20U2956
3- STEPHANE ROYLEX NKOLO	21T2588
4- TCHAMI TAMEN SORELLE	20U2855
5- KOUAM NOUBISSI SERAPHIN BRICE	21T2432

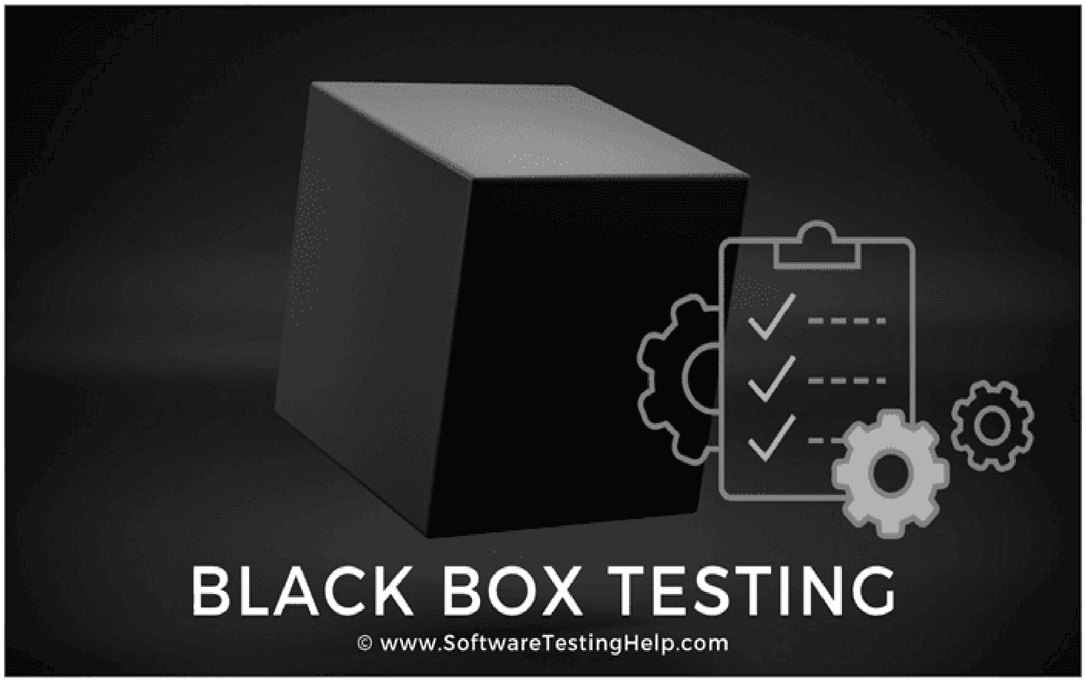
08 MAI 2024

SUPERVISED BY ***DR. KIMBI XAVIERA***

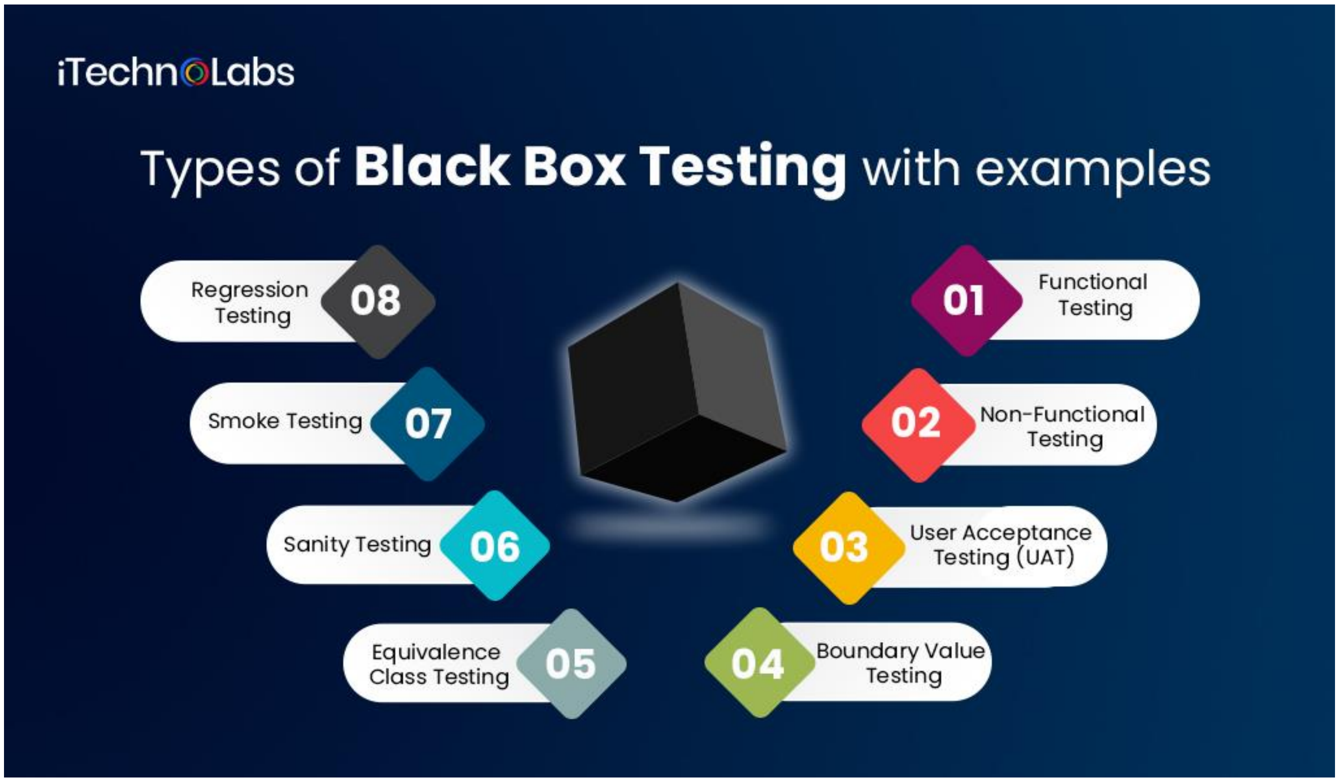
Contents

Introduction

In the complex world of software development, where every line of code is meticulously designed to meet specific needs, a crucial challenge persists: how to ensure that software works as expected, even without knowing its innards?
Black box testing, sometimes called specification-based testing, is emerging as an answer to this question. This methodology examines the functionality of an application without looking at its internal structures or functioning. It allows testing teams to explore the software without any prior knowledge of its internal architecture.
Imagine a world where testers find themselves in the position of an ordinary user, unaware of source code intricacies or design details. It is in this context that black box testing proves to be an essential approach.
Like an end user, testers provide input, perform actions, and observe results, all while remaining blind to implementation details. The ultimate goal of this approach is to deliver an authentic user experience, identifying possible issues or unexpected behaviors that could compromise that experience.
Black box testing can be applied to virtually all levels of software testing: unit, integration, system, and acceptance. They are also used as a penetration testing method, in which an ethical hacker simulates an external hack or cyberwarfare attack without knowledge of the system being attacked.
By deliberately separating testing teams from development teams, this methodology promotes objective and unbiased evaluation, thereby strengthening software quality and reliability in an ever-changing technology landscape.



I. *Types of Black box testing*


iTechnoLabs
Types of Black Box Testing with examples
01 Functional Testing
02 Non-Functional Testing
03 User Acceptance Testing (UAT)
04 Boundary Value Testing
05 Equivalence Class Testing
06 Sanity Testing
07 Smoke Testing
08 Regression Testing

Black box testing is a methodology of performing tests. These tests can be designed to accomplish a few different goals, including:

<u>Functional Testing</u>: Functional testing is intended to validate that an application does what it is supposed to do. For example, functional tests may test an application's authentication mechanism to check that legitimate users can authenticate successfully while invalid login attempts are rejected. Common types of functional testing include sanity checks, integration testing, and system testing.

<u>Non-Functional Testing</u>: Non-functional testing evaluates how well an application performs its core functions. Examples of tests include performance, usability, scalability, and security testing.

<u>Regression Testing</u>: Regression testing is designed to ensure that a change to an application does not break functionality. For example, regression testing should be performed after patching a vulnerability in an application to ensure the patch has not caused the application to fail functional or non-functional tests.

<u>Equivalence Class Testing</u>: An application may follow the same control flow for certain types of inputs. For example, an application that should only be accessible to adults may terminate if a user enters an age under 18 or a tool with a limited-service area may terminate for country or postal codes outside of that area. With equivalence class testing, testers identify these classes that produce the same results and only test for one value within that class.

<u>Boundary Value Evaluation</u>: Boundary values are inputs where an application's changes from one control flow to another. For example, the ages 17 and 18 are boundary values for adulthood since a 17-

year-old may be rejected by an application, while an 18-year-old would be accepted. Boundary value evaluation tests these inputs to ensure that the system is properly handling these edge cases.
Decision Table Testing: An application may be designed to make decisions based on a combination of inputs. For example, users over the age of 18 and living within a particular area may be able to access an application. Decision table testing involves enumerating each combination of inputs and its expected outcomes and developing a test case to validate each combination.

State Transition Evaluation: An application may be designed to change state under certain conditions, such as locking a user's account after a certain number of failed authentication attempts. State transition evaluation involves identifying these situations and developing test cases to validate them.
Error Checking: This form of evaluation tests for common errors that a developer may have made when creating an application. This often revolves around input sanitization and ensuring that assumptions about an input are enforced. For example, testers may check to see if developers properly handled an input of zero in a numeric field or restricted the character set for a name to the letters and symbols that can appear in a name.

II. Features of black box testing

Independent testing: Black box testing is performed by testers who are not involved in the development of the application, which helps to ensure that testing is unbiased and impartial.

Testing from a user's perspective: Black box testing is conducted from the perspective of an end user, which helps to ensure that the application meets user requirements and is easy to use.

No knowledge of internal code: Testers performing black box testing do not have access to the application's internal code, which allows them to focus on testing the application's external behaviour and functionality.

Requirements-based testing: Black box testing is typically based on the application's requirements, which helps to ensure that the application meets the required specifications.

Different testing techniques: Black box testing can be performed using various testing techniques, such as functional testing, usability testing, acceptance testing, and regression testing.

Easy to automate: Black box testing is easy to automate using various automation tools, which helps to reduce the overall testing time and effort.

Scalability: Black box testing can be scaled up or down depending on the size and complexity of the application being tested.

Limited knowledge of application: Testers performing black box testing have limited knowledge of the application being tested, which helps to ensure that testing is more representative of how the end users will interact with the application.

III. Advantages

The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.

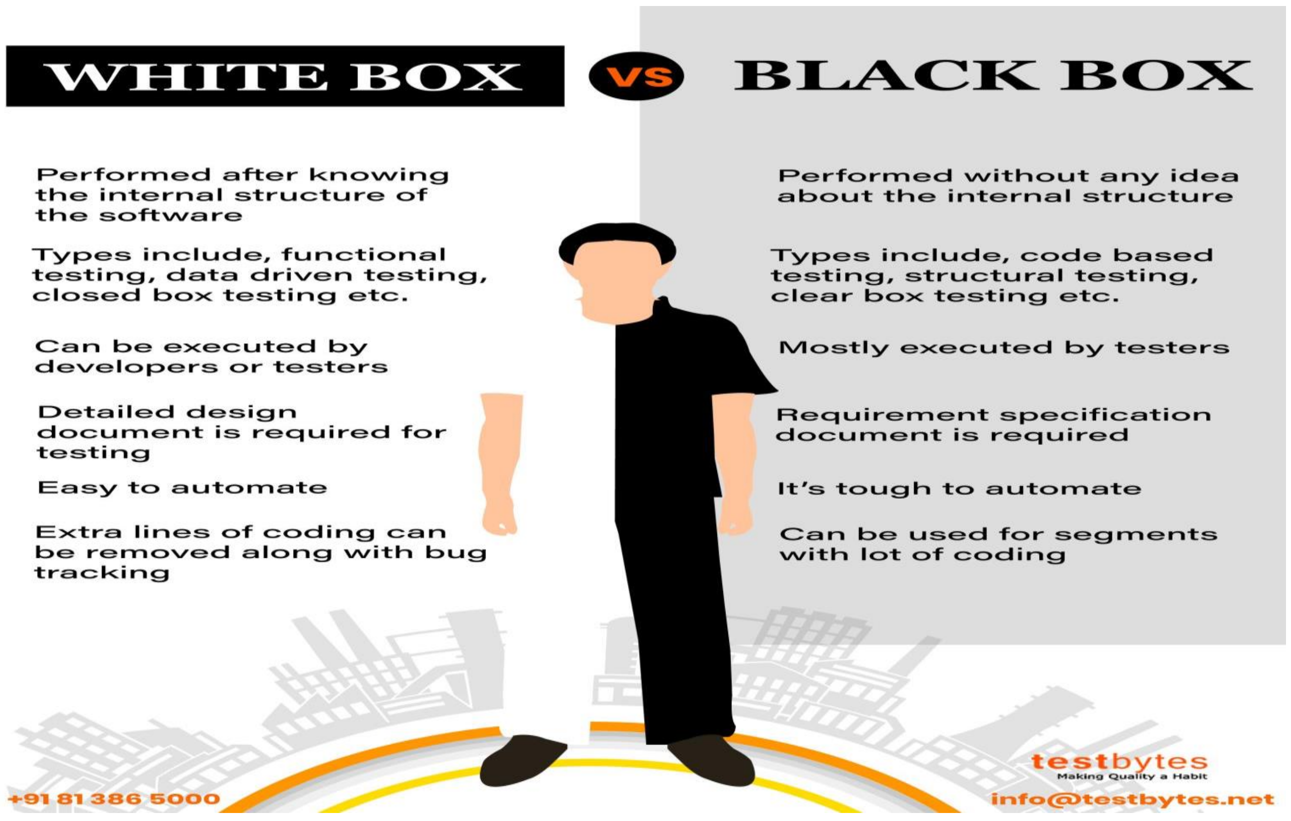
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications.

IV. Disadvantages

There is a possibility of repeating the same tests while implementing the testing process.
Without clear functional specifications, test cases are difficult to implement.

- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

V. Black Box vs White Box Testing



WHITE BOX	BLACK BOX
Performed after knowing the internal structure of the software	Performed without any idea about the internal structure
Types include, functional testing, data driven testing, closed box testing etc.	Types include, code based testing, structural testing, clear box testing etc.
Can be executed by developers or testers	Mostly executed by testers
Detailed design document is required for testing	Requirement specification document is required
Easy to automate	It's tough to automate
Extra lines of coding can be removed along with bug tracking	Can be used for segments with lot of coding

While black box testing is named for the fact that the tester has no internal knowledge of the application (i.e. it's a "black box"), a white box evaluation takes the opposite approach. Some of the key differences between black box and white box testing include:

Black Box Testing: The tester interacts with the application and attempts to validate that an application meets functional and non-functional requirements and specifications. The lack of internal knowledge can make these tests more time-consuming and may cause vulnerabilities in unvisited code paths to go undetected. However, it has the advantage of being language and platform-agnostic.

White Box Testing: As opposed to black box testing, white box evaluations are performed with full knowledge of an application's internals, including access to the source code. White box testing offers better test coverage than black box testing since all code can be evaluated. However, it requires expertise with the language in which the code was developed.
Black box and white box testing represent two extremes in how testing can be performed. Gray box testing falls in between. In a gray box evaluation, the tester has partial knowledge of the system's internals, which can help to guide the evaluation. Runtime application self-protection (RASP) is a security tool that falls into the gray box testing category.

Whitebox Testing	**Blackbox Testing**
White box testing (also known as clear box testing or structural testing) focuses on examining the internal code and logic of the software. Testers have access to the source code and use this knowledge to design test cases that target specific code paths and conditions.	Black box testing, on the other hand, focuses on the external behavior of the software without having access to the internal code. Testers design test cases based on the software's requirements and specifications, testing how the system responds to various inputs.
White box testing requires testers to have an understanding of the internal code and the software's architecture. This knowledge is essential to identify potential code issues and design tests to exercise specific code branches.	Black box testing does not require knowledge of the internal code, making it suitable for non-technical testers or those who do not have access to the source code.
White box testing uses techniques such as code coverage analysis, statement coverage, branch coverage, and path coverage to design test cases that ensure maximum code coverage and exercise all possible code paths.	Black box testing uses techniques like equivalence partitioning, boundary value analysis, decision tables, and state transition testing to design test cases based on the software's requirements, input ranges, and expected behavior.
White box testing is typically performed at the unit testing and integration testing levels, where the internal code can be accessed and tested.	Black box testing can be applied at various testing levels, including unit testing, integration testing, system testing, and acceptance testing, focusing on the external functionality of the software.

White box testing aims to verify the correctness of the internal code, identify logical errors, and ensure that all code paths are tested.	Black box testing aims to validate the software's functionality, ensure it meets the specified requirements, and assess how it behaves from an end-user perspective.
In white box testing, testers need to have programming skills and an understanding of the codebase to design and execute effective tests.	In black box testing, testers do not need programming knowledge, and it can be performed by non-technical team members.

VI. Tools and Frameworks used to perform Black Box Testing

There are several black box testing tools available that can assist testers in automating and managing the testing process for software applications. These tools help with creating and executing test cases, capturing test results, and generating reports.

Some popular black box testing tools include:

1. Selenium

Selenium is commonly used for black box testing, particularly for web applications. Selenium is an open-source testing framework that allows testers to automate the testing of web browsers, making it a valuable tool for performing black box testing on web-based systems. It interacts with web elements on the user interface, simulating real user interactions and validating the functionality of the application without accessing its internal code.

2. Appium

Appium is another popular tool that is often used for black box testing, particularly for mobile applications. Appium is an open-source test automation framework that allows testers to automate the testing of native, hybrid, and mobile web applications on both Android and iOS devices. It enables black box testing of mobile apps without accessing the internal code.

Pro Tip :It is advised to perform Selenium and Appium Tests on real device cloud to obtain more accurate test results.

3. Cypress

Cypress is a powerful test automation framework primarily used for front-end testing, including end-to-end (E2E) testing and user interface (UI) testing. While Cypress is more commonly associated with white box testing due to its ability to access and control the application's internal code, it can also be used for black box testing to some extent.

While Cypress may provide some black box testing capabilities, its real strength lies in the combination of white box and black box testing. For instance, testers can use Cypress to conduct E2E tests and then complement it with other black box testing techniques like exploratory testing or usability testing.

4. Load Runner

LoadRunner is primarily known as a performance testing tool, and its core focus is on testing the performance, scalability, and reliability of applications under different load conditions. While LoadRunner is not typically used as a dedicated black box testing tool, it can still be employed to perform some aspects of black box testing in specific scenarios, for e.g.

Load Testing with Real User Scenarios: LoadRunner can simulate real user scenarios and interactions with the application. In this sense, it acts as a black box, not having direct access to the application's internal code.
User Experience Testing: By conducting load tests with multiple virtual users, LoadRunner can help assess the overall user experience. It measures the application's response times, resource utilization, and other performance metrics, simulating real-world scenarios from the end-user perspective.

5. SoapUI

SoapUI is primarily known as an API testing tool, and its main focus is on testing the functionality and behavior of APIs (Web services). As such, SoapUI is well-suited for black box testing of APIs, ensuring that they meet the specified requirements without needing access to the underlying code.

Here's how SoapUI can be used for black box testing of APIs:

Functional Testing: SoapUI allows testers to create test cases that simulate API requests and responses. Testers can validate if the API functions correctly based on the expected results without knowing the internal implementation.
Input Validation: Testers can use SoapUI to check how the API handles different types of inputs and whether it provides the appropriate responses, such as error messages for invalid data.
Boundary Value Analysis: SoapUI allows testers to test API responses with boundary values to verify if the API behaves correctly at the edges of the input range.
For comprehensive black box testing, especially when dealing with end-to-end testing of web applications, it is recommended to use dedicated black box testing tools like Selenium or Cypress in combination with SoapUI for API testing. This combination allows for a more complete testing approach, covering both the functionality of the APIs and the user interface interactions of the application.

VII. Best Practices for Black Box Testing

Effective black box testing requires careful planning, thorough test case design, and meticulous execution. Here are some best practices to ensure successful black box testing:

Requirement Analysis: Start by thoroughly understanding the software's requirements and specifications. Clear and well-defined requirements will guide the creation of meaningful test cases.
Test Planning: Develop a comprehensive test plan that outlines the testing scope, objectives, testing levels, resources, and timelines. This will serve as a roadmap for the testing process.
Test Case Design Techniques: Utilize various test case design techniques like equivalence partitioning, boundary value analysis, decision tables, and state transition testing to ensure comprehensive test coverage.

Test Data Management: Prepare relevant and diverse test data to cover various scenarios. Validate both valid and invalid inputs to assess the software's response.

Positive and Negative Testing: Include test cases for both positive scenarios (valid inputs with expected outcomes) and negative scenarios (invalid inputs with appropriate error handling).

Usability Testing: Focus on testing the user interface and overall user experience. Verify that the application is user-friendly, consistent, and easy to navigate.

Regression Testing: As changes are made to the software, perform regression testing to ensure that new updates or fixes do not introduce new defects or impact existing functionality.

Boundary Value Analysis: Test the software's behavior around the boundaries of input ranges to identify potential issues with boundary conditions.

Error Localization and Reporting: Clearly document and report any defects or issues discovered during testing, including detailed steps to reproduce the problem and information on the test environment.

Test Automation: Automate repetitive and time-consuming test cases to improve testing efficiency and repeatability. Automation helps in running tests more frequently and consistently.

By following these best practices, testers can conduct thorough and effective black box testing, identifying and resolving defects, and ensuring that the software meets the desired quality standards and user requirements.

VIII. Black Box Testing and Software Development Life Cycle (SDLC)

Black box testing has its own life cycle called Software Testing Life Cycle (STLC) and it is relative to every stage of Software Development Life Cycle of Software Engineering.

Requirement – This is the initial stage of SDLC and in this stage, a requirement is gathered. Software testers also take part in this stage.
Test Planning & Analysis – Testing Types applicable to the project are determined. A Test Plan is created which determines possible project risks and their mitigation.
Design – In this stage Test cases/scripts are created on the basis of software requirement documents
Test Execution – In this stage Test Cases prepared are executed. Bugs if any are fixed and re-tested.

Conclusion

These are some of the basic points regarding Black box testing and the overview of its techniques and methods.

As it is not possible to test everything with human involvement with 100 percent accuracy, if the above-mentioned techniques and methods are used effectively, then it will definitely improve the quality of the system.

To conclude, this is a very helpful method to verify the functionality of the system and identify most of the defects.