## What is Black Box testing?

In Black-box testing, a tester doesn't have any information about the internal working of the software system. Black box testing is a high level of testing that focuses on the behavior of the software. It involves testing from an external or end-user perspective. Black box testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance.

## What is White Box testing?

White-box testing is a testing technique which checks the internal functioning of the system. In this method, testing is based on coverage of code statements, branches, paths or conditions. White-Box testing is considered as low-level testing. It is also called glass box, transparent box, clear box or code base testing. The white-box Testing method assumes that the path of the logic in a unit or program is known.

**KEY DIFFERENCE**

* In Black Box, testing is done without the knowledge of the internal structure of program or application whereas in White Box, testing is done with knowledge of the internal structure of program.
* Black Box test doesn’t require programming knowledge whereas the White Box test requires programming knowledge.
* Black Box testing has the main goal to test the behavior of the software whereas White Box testing has the main goal to test the internal operation of the system.
* Black Box testing is focused on external or end-user perspective whereas White Box testing is focused on code structure, conditions, paths and branches.
* Black Box test provides low granularity reports whereas the White Box test provides high granularity reports.
* Black Box testing is a not time-consuming process whereas White Box testing is a time-consuming process.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Black Box testing** | **White Box testing** |
| **Definition** | It is a testing approach which is used to test the software without the knowledge of the internal structure of program or application. | It is a testing approach in which internal structure is known to the tester. |
| **Alias** | It also knowns as data-driven, box testing, data-, and functional testing. | It is also called structural testing, clear box testing, code-based testing, or glass box testing. |
| **Base of Testing** | Testing is based on external expectations; internal behavior of the application is unknown. | Internal working is known, and the tester can test accordingly. |
| **Usage** | This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing. | Testing is best suited for a lower level of testing like Unit Testing, Integration testing. |
| **Programming knowledge** | Programming knowledge is not needed to perform Black Box testing. | Programming knowledge is required to perform White Box testing. |
| **Implementation knowledge** | Implementation knowledge is not requiring doing Black Box testing. | Complete understanding needs to implement WhiteBox testing. |
| **Automation** | Test and programmer are dependent on each other, so it is tough to automate. | White Box testing is easy to automate. |
| **Objective** | The main objective of this testing is to check what functionality of the system under test. | The main objective of White Box testing is done to check the quality of the code. |
| **Basis for test cases** | Testing can start after preparing requirement specification document. | Testing can start after preparing for Detail design document. |
| **Tested by** | Performed by the end user, developer, and tester. | Usually done by tester and developers. |
| **Granularity** | Granularity is low. | Granularity is high. |
| **Testing method** | It is based on trial and error method. | Data domain and internal boundaries can be tested. |
| **Time** | It is less exhaustive and time-consuming. | Exhaustive and time-consuming method. |
| **Algorithm test** | Not the best method for algorithm testing. | Best suited for algorithm testing. |
| **Code Access** | Code access is not required for Black Box Testing. | White box testing requires code access. Thereby, the code could be stolen if testing is outsourced. |
| **Benefit** | Well suited and efficient for large code segments. | It allows removing the extra lines of code, which can bring in hidden defects. |
| **Skill level** | Low skilled testers can test the application with no knowledge of the implementation of programming language or operating system. | Need an expert tester with vast experience to perform white box testing. |
| **Techniques** | Equivalence partitioning is Black box testing technique is used for Blackbox testing. | Statement Coverage, Branch coverage, and Path coverage are White Box testing technique. |
|  |  |
| Equivalence partitioning divides input values into valid and invalid partitions and selecting corresponding values from each partition of the test data. | Statement Coverage validates whether every line of the code is executed at least once. |
|  |  |
| Boundary value analysis | Branch coverage validates whether each branch is executed at least once |
|  |  |
| checks boundaries for input values. | Path coverage method tests all the paths of the program. |
| **Drawbacks** | Update to automation test script is essential if you to modify application frequently. | Automated test cases can become useless if the code base is rapidly changing. |

## What Is Accessibility Testing?

Accessibility testing is the practice of making your web and mobile apps usable to as many people as possible. It makes apps accessible to those with disabilities, such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

Accessibility testing works best when incorporated into your testing strategies — don’t let it be an afterthought. Align it with your test cycle and sync your results all in one place with Perfecto’s[test reporting](https://www.perfecto.io/platform/analytics-reporting).

## What is Functional Testing?

Functional testing is the process of validating functionality of a software application. Pass or fail is the result of a functional test, because either a feature works as designed or it does not.

The purpose of functional testing is to validate that the requirements of the software application have been met. It is important because functional testing assesses an application’s fitness to be released to end users. While software engineering has evolved in the past decade, functional testing remains a core part of quality testing.

## Why You Need Interactive Testing

While automation is certainly helpful for DevOps teams, some testing just needs to be done manually.

Repetitive, high-frequency tests can easily be automated. But tests should not be automated if they occur infrequently, are high risk, or are subject to change.

Instances when you’d want live testing instead of automated testing:

* UX and exploratory testing.
* Complex test scenarios that are not efficient and sometimes not feasible to automate.
* Test scenarios that are only being validated once in a while.

## What Is Nonfunctional Testing?

Nonfunctional testing verifies how applications work by examining things like performance, accessibility, and UX.

## Difference Between Functional and Nonfunctional Testing

The difference between functional and nonfunctional testing is what they test.

Functional testing ensures that functions and features of the application work properly. Nonfunctional testing examines other aspects of how well the application works.

Functional testing tests the functionality of an app. Nonfunctional testing tests the performance of these functions.

Some examples of functional testing include unit testing, integration testing, API testing, exploratory testing, and critical business flows testing, These all test functional aspects of the website or mobile app.

## Why You Need Nonfunctional Testing

Nonfunctional testing is just as critical as functional testing. And because teams need to conduct a mix of different [types of testing](https://www.perfecto.io/resources/types-of-testing), you need to be doing both.

Nonfunctional testing makes applications more usable and more reliable. Unfortunately, it can often be rushed in an effort to meet release deadlines.

When nonfunctional testing is overlooked, performance and UX defects can leave users with a bad experience and cause brand damage. Worse, applications could crash with an influx of users. Accessibility defects can result in compliance fines. And their security could be at risk.

## What Are Nonfunctional Testing Types?

Nonfunctional testing is an umbrella term. There are many nonfunctional testing types. Here are a few.

* **Accessibility testing** — Tests how usable the app is to users with disabilities, such as vision impairment.
* **Availability testing** — Tests how often the app is accessible and readily available for use.
* **Compliance testing** — Tests whether an app meets specified requirements or regulations.
* **Configuration testing** — Tests an app against software and hardware variations.
* **Disaster recovery testing** — Tests recovery of business-critical applications in emergency situations.
* **Endurance testing** — Tests an app under a heavy load over an extended period of time.
* **Failover testing** — Tests an app’s backup system in the event of a system failure.
* **Geolocation testing** — Tests location-based scenarios on an app.
* **Internationalization testing** — Tests if an app can adapt to regional languages and other factors based on location.
* **Load testing** — Tests an app’s performance under peak conditions.
* **Maintainability testing** — Tests the app’s ability to update.
* **Performance testing** — Tests the speed and responsiveness of an app under various conditions.
* **Portability testing** — Tests how an app transfers from one software or operational environment to another.
* **Resilience testing** — Tests an app’s ability to perform under stressed conditions.
* **Security testing** — Tests an app’s security mechanisms to reveal vulnerabilities.
* **Scalability testing** — Tests an app’s ability to scale up or down as user requests vary.
* **Stress testing** — Tests an app’s stability under heavy loads or extreme conditions.
* **Usability testing** — Tests an app’s ease of use.

## The Risks of Testing Too Late

The traditional approach towards non-functional testing means it occurs at the end of the software delivery process, sometimes even extending beyond. Too often, nonfunctional testing is either left to the end of the cycle or done only partially. Or, it’s outsourced externally, where it is performed manually due to a lack of time and automation abilities.

In these cases, critical tests don’t run in time to fully guarantee the delivery readiness of upcoming releases. This risks brand damage, compliance issues, and worse.

## Shifting Nonfunctional Testing Left

What can be done to fix this?

Modern testing frameworks and [cloud-based solutions](https://www.perfecto.io/blog/why-devops-teams-need-cloud-based-solutions) allow earlier [testing in the SDLC](https://www.perfecto.io/blog/right-testing-tool-each-phase-sdlc). These allow for identification and resolution of nonfunctional testing defects. Without nonfunctional testing early in the cycle, these defects have the potential to delay release. Nonfunctional testing thus reduces costs, unnecessary effort, and risk.

Early testing also smooths the transition from development to functional and nonfunctional testing.

Technology is only one half of the equation though. A cultural commitment to Agile principles is also needed. Teams need to constantly monitor and fine tune their tests so that they are aligned, well-perceived, and trusted.

## Bottom Line

Nonfunctional testing is a critical part of any testing strategy. But it has to be able to scale to your testing needs.

Try nonfunctional testing with Perfecto’s cloud testing platform.

Execute your tests with real user conditions against devices and browsers in the cloud. Perfecto makes it easy to scale and analyze tests.

## Regression Testing Is Essential

Small changes can have big consequences. That’s why teams do regression testing. Regression testing makes sure that code changes don’t break anything. It catches bugs early on in new builds. Without regression testing, it’s more difficult, time intensive, and expensive to find defects.

## What Is Performance Testing?

Performance testing is a type of [non-functional](https://www.perfecto.io/blog/what-is-non-functional-testing) testing. It tests the quality of an app under varying capacities.

<https://www.perfecto.io/blog/single-user-performance-testing>

**Manual testing** is the most hands-on type of testing and is employed by every team at some point. Of course, in today’s fast-paced software development lifecycle, manual testing is tough to scale.

**Automated testing**uses test scripts and specialized tools to automate the process of software testing.

**Continuous testing** goes even further, applying the principles of automated testing in a scaled, continuous manner to achieve the most reliable test coverage for an enterprise. Keep reading to learn more about the differences between [automated testing vs. manual testing](https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing) and how continuous testing fits in.

https://www.perfecto.io/resources/types-of-testing

JUnit provides basic and advanced concepts of **unit testing in java** with examples.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test own code.

It is one of the unit testing framework.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

The **org.junit** package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

Types of unit testing

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

1) Manual Testing

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

2) Automated Testing

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

#### Annotations for Junit testing

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

**@Test** annotation specifies that method is the test method.

**@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

**@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.

**@Before** annotation specifies that method will be invoked before each test.

**@After** annotation specifies that method will be invoked after each test.

**@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

## Assert class

The org.junit.Assert class provides methods to assert the program logic.

#### Methods of Assert class

The common methods of Assert class are as follows:

1. **void assertEquals(boolean expected,boolean actual)**: checks that two primitives/objects are equal. It is overloaded.
2. **void assertTrue(boolean condition)**: checks that a condition is true.
3. **void assertFalse(boolean condition)**: checks that a condition is false.
4. **void assertNull(Object obj)**: checks that object is null.
5. **void assertNotNull(Object obj)**: checks that object is not null.

#### Required jar files

You need to load **junit4.jar** and **hamcrest-core.jar** files.

[download the junit jar files](https://www.javatpoint.com/src/junit/junit4jars.zip)