Software Testing Methods

**SOFTWARE TESTING METHODS** listed here are the major methods used while conducting various Software Testing Types during various Software Testing Levels:

|  |  |
| --- | --- |
| **Method** | **Summary** |
| Black Box Testing | A software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional. Test design techniques include *Equivalence partitioning, Boundary Value Analysis, Cause-Effect Graphing.* |
| White Box Testing | A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Test design techniques include *Control flow testing, Data flow testing, Branch testing, Path testing.* |
| Gray Box Testing | A software testing method which is a combination of Black Box Testing method and White Box Testing method. |
| Agile Testing | A method of software testing that follows the principles of agile software development. |
| Ad Hoc Testing | A method of software testing without any planning and documentation. |

Black Box Testing

**BLACK BOX TESTING**, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

Definition by ISTQB

* **black box testing:**Testing, either functional or non-functional, without reference to the internal structure of the component or system.
* **black box test design technique:**Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

Example

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Levels Applicable To

Black Box Testing method is applicable to the following levels of software testing:

* Integration Testing
* System Testing
* Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

Techniques

Following are some techniques that can be used for designing black box tests.

* *Equivalence Partitioning:*It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
* *Boundary Value Analysis:*It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
* *Cause-Effect Graphing:*It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

Advantages

* Tests are done from a user’s point of view and will help in exposing discrepancies in the specifications.
* Tester need not know programming languages or how the software has been implemented.
* Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
* Test cases can be designed as soon as the specifications are complete.

Disadvantages

* Only a small number of possible inputs can be tested and many program paths will be left untested.
* Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
* Tests can be redundant if the software designer/developer has already run a test case.
* Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

Integration Testing

**INTEGRATION TESTING** is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.



Definition by ISTQB

* **integration testing:**Testing performed to expose defects in the interfaces and in the
interactions between integrated components or systems. See also c*omponent integration
testing, system integration testing*.
* **component integration testing:**Testing performed to expose defects in the interfaces and
interaction between integrated components.
* **system integration testing:**Testing the integration of systems and packages; testing
interfaces to external organizations (e.g. Electronic Data Interchange, Internet).

Analogy

During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. For example, whether the cap fits into the body or not.

Method

Any of Black Box Testing, White Box Testing and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.

Tasks

* Integration Test Plan
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Integration Test Cases/Scripts
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Integration Test
	+ Perform

**When is Integration Testing performed?**

Integration Testing is the second level of testing performed after Unit Testing and before System Testing.

**Who performs Integration Testing?**

Developers themselves or independent testers perform Integration Testing.

Approaches

* *Big Bang* is an approach to Integration Testing where all or most of the units are combined together and tested at one go. This approach is taken when the testing team receives the entire software in a bundle. So what is the difference between Big Bang Integration Testing and System Testing? Well, the former tests only the interactions between the units while the latter tests the entire system.
* *Top Down* is an approach to Integration Testing where top-level units are tested first and lower level units are tested step by step after that. This approach is taken when top-down development approach is followed. Test Stubs are needed to simulate lower level units which may not be available during the initial phases.
* *Bottom Up* is an approach to Integration Testing where bottom level units are tested first and upper-level units step by step after that. This approach is taken when bottom-up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases.
* *Sandwich/Hybrid* is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.

Tips

* Ensure that you have a proper Detail Design document where interactions between each unit are clearly defined. In fact, you will not be able to perform Integration Testing without this information.
* Ensure that you have a robust Software Configuration Management system in place. Or else, you will have a tough time tracking the right version of each unit, especially if the number of units to be integrated is huge.
* Make sure that each unit is unit tested before you start Integration Testing.
* As far as possible, automate your tests, especially when you use the Top Down or Bottom Up approach, since regression testing is important each time you integrate a unit, and manual regression testing can be inefficient.

System Testing

**SYSTEM TESTING**is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements.



Definition by ISTQB

* **system testing:**The process of testing an integrated system to verify that it meets specified requirements.

Analogy

During the process of manufacturing a ballpoint pen, the cap, the body, the tail, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. When the complete pen is integrated, System Testing is performed.

Method

Usually, Black Box Testing method is used.

Tasks

* System Test Plan
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* System Test Cases
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* System Test
	+ Perform

**When is it performed?**

System Testing is the third level of software testing performed after Integration Testing and before Acceptance Testing.

**Who performs it?**

Normally, independent Testers perform System Testing.

Acceptance Testing

**ACCEPTANCE TESTING**is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.



Definition by ISTQB

* **acceptance testing:**Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

Analogy

During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. When the complete pen is integrated, System Testing is performed. Once System Testing is complete, Acceptance Testing is performed so as to confirm that the ballpoint pen is ready to be made available to the end-users.

Method

Usually, Black Box Testing method is used in Acceptance Testing. Testing does not normally follow a strict procedure and is not scripted but is rather ad-hoc.

Tasks

* Acceptance Test Plan
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Acceptance Test Cases/Checklist
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Acceptance Test
	+ Perform

**When is it performed?**

Acceptance Testing is the fourth and last level of software testing performed after System Testing and before making the system available for actual use.

**Who performs it?**

* *Internal Acceptance Testing*(Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.
* *External Acceptance Testing*is performed by people who are not employees of the organization that developed the software.
	+ *Customer Acceptance Testing*is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software. [This is in the case of the software not being owned by the organization that developed it.]
	+ *User Acceptance Testing*(Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers’ customers.

White Box Testing

**WHITE BOX TESTING**(also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.

Definition by ISTQB

* **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
* **white-box test design technique:**Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Example

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Levels Applicable To

White Box Testing method is applicable to the following levels of software testing:

* Unit Testing: For testing paths within a unit.
* Integration Testing: For testing paths between units.
* System Testing: For testing paths between subsystems.

However, it is mainly applied to Unit Testing.

Advantages

* Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
* Testing is more thorough, with the possibility of covering most paths.

Disadvantages

* Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
* Test script maintenance can be a burden if the implementation changes too frequently.
* Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.

Unit Testing

**UNIT TESTING** is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.



Definition by ISTQB

* **unit testing:** See *component testing.*
* **component testing:**The testing of individual software components.

Unit Testing Method

It is performed by using the White Box Testing method.

**When is it performed?**

Unit Testing is the first level of software testing and is performed prior to Integration Testing.

**Who performs it?**

It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers.

Unit Testing Tasks

* Unit Test Plan
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Unit Test Cases/Scripts
	+ Prepare
	+ Review
	+ Rework
	+ Baseline
* Unit Test
	+ Perform

Unit Testing Benefits

* Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
* Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
* Development is faster. How? If you do not have unit testing in place, you write your code and perform that fuzzy ‘developer test’ (You set some breakpoints, fire up the GUI, provide a few inputs that hopefully hit your code and hope that you are all set.) But, if you have unit testing in place, you write the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; You need not fire up the GUI and provide all those inputs. And, of course, unit tests are more reliable than ‘developer tests’. Development is faster in the long run too. How? The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.
* The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.
* Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be scanned.
* Codes are more reliable. Why? I think there is no need to explain this to a sane person.

Unit Testing Tips

* Find a tool/framework for your language.
* Do not create test cases for everything. Instead, focus on the tests that impact the behavior of the system.
* Isolate the development environment from the test environment.
* Use test data that is close to that of production.
* Before fixing a defect, write a test that exposes the defect. Why? First, you will later be able to catch the defect if you do not fix it properly. Second, your test suite is now more comprehensive. Third, you will most probably be too lazy to write the test after you have already fixed the defect.
* Write test cases that are independent of each other. For example, if a class depends on a database, do not write a case that interacts with the database to test the class. Instead, create an abstract interface around that database connection and implement that interface with a mock object.
* Aim at covering all paths through the unit. Pay particular attention to loop conditions.
* Make sure you are using a version control system to keep track of your test scripts.
* In addition to writing cases to verify the behavior, write cases to ensure the performance of the code.
* Perform unit tests continuously and frequently.

One more reason

Let’s say you have a program comprising of two units and the only test you perform is system testing. [You skip unit and integration testing.] During testing, you find a bug. Now, how will you determine the cause of the problem?

* Is the bug due to an error in unit 1?
* Is the bug due to an error in unit 2?
* Is the bug due to errors in both units?
* Is the bug due to an error in the interface between the units?
* Is the bug due to an error in the test or test case?

Unit testing is often neglected but it is, in fact, the most important level of testing.

Differences Between Black Box Testing and White Box Testing

The Differences Between Black Box Testing and White Box Testing are listed below.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Black Box Testing** | **White Box Testing** |
| *Definition* | Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester | White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. |
| *Levels Applicable To* | Mainly applicable to higher levels of testing:Acceptance TestingSystem Testing | Mainly applicable to lower levels of testing:Unit TestingIntegration Testing |
| *Responsibility* | Generally, independent Software Testers | Generally, Software Developers |
| *Programming Knowledge* | Not Required | Required |
| *Implementation Knowledge* | Not Required | Required |
| *Basis for Test Cases* | Requirement Specifications | Detail Design |

# Gray Box Testing

**GRAY BOX TESTING**is a software testing method which is a combination of Black Box Testing method and White Box Testing method. In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure is known. In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray/semi-transparent box; inside which one can partially see.

## Example

An example of Gray Box Testing would be when the codes for two units/modules are studied (White Box Testing method) for designing test cases and actual tests are conducted using the exposed interfaces (Black Box Testing method).

## Levels Applicable To

Though Gray Box Testing method may be used in other levels of testing, it is primarily used in Integration Testing.

## Spelling

Note that Gray is also spelled as Grey. Hence Grey Box Testing and Gray Box Testing mean the same.

**What do you verify in White Box Testing?**

White box testing involves the testing of the software code for the following:

* Internal security holes
* Broken or poorly structured paths in the coding processes
* The flow of specific inputs through the code
* Expected output
* The functionality of conditional loops
* Testing of each statement, object, and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

Click here if the video is not accessible

**How do you perform White Box Testing?**

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

**STEP 1) UNDERSTAND THE SOURCE CODE**

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

**Step 2) CREATE TEST CASES AND EXECUTE**

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

**WhiteBox Testing Example**

Consider the following piece of code

Printme (int a, int b) { ------------ Printme is a function

 int result = a+ b;

 If (result> 0)

 Print ("Positive", result)

 Else

 Print ("Negative", result)

 } ----------- End of the source code

The goal of WhiteBox testing is to verify all the decision branches, loops, statements in the code.

To exercise the statements in the above code, WhiteBox test cases would be

* A = 1, B = 1
* A = -1, B = -3

**White Box Testing Techniques**

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques

**Statement Coverage**:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

**Branch Coverage -**This technique checks every possible path (if-else and other conditional loops) of a software application.

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.

To learn more in detail refer this article https://www.guru99.com/code-coverage.html

**Types of White Box Testing**

*White box testing*encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below --

* **Unit Testing:**It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.
* **Testing for Memory Leaks**: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

Apart from above, a few testing types are part of both black box and white box testing. They are listed as below

* **White Box Penetration Testing:** In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on.  The aim is to attack the code from several angles to expose security threats
* **White Box Mutation Testing**: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution.

**White Box Testing Tools**

Below is a list of top white box testing tools.

* Veracode
* EclEmma
* RCUNIT
* NUnit
* JSUnit
* JUnit
* CppUnit

**Advantages of White Box Testing**

* Code optimization by finding hidden errors.
* White box tests cases can be easily automated.
* Testing is more thorough as all code paths are usually covered.
* Testing can start early in SDLC even if GUI is not available.

**Disadvantages of WhiteBox Testing**

* White box testing can be quite complex and expensive.
* Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
* White box testing requires professional resources, with a detailed understanding of programming and implementation.
* White-box testing is time-consuming, bigger programming applications take the time to test fully.

**Ending Notes:**

* White box testing can be quite complex. The complexity involved has a lot to do with the application being tested. A small application that performs a single simple operation could be white box tested in few minutes, while larger programming applications take days, weeks and even longer to fully test.
* White box testing should be done on a software application as it is being developed after it is written and again after each modification

**Working process of white box testing:**

* **Input:** Requirements, Functional specifications, design documents, source code.
* **Processing:** Performing risk analysis for guiding through the entire process.
* **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
* **Output:** Preparing final report of the entire testing process.

**Testing techniques:**

* **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



*Statement Coverage Example*

* **Branch Coverge:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



*4 test cases required such that all branches of all decisions are covered, i.e, all edges of flowchart are covered*

* **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:
	1. READ X, Y
	2. IF(X == 0 || Y == 0)
	3. PRINT ‘0’

In this example, there are 2 conditions: X == 0 and Y == 0. Now, test these conditions get TRUE and FALSE as their values. One possible example would be:

* 1. #TC1 – X = 0, Y = 55
	2. #TC2 – X = 5, Y = 0
* **Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example:
	1. READ X, Y
	2. IF(X == 0 || Y == 0)
	3. PRINT ‘0’
	4. #TC1: X = 0, Y = 0
	5. #TC2: X = 0, Y = 5
	6. #TC3: X = 55, Y = 0
	7. #TC4: X = 55, Y = 5

Hence, four test cases required for two individual conditions.
Similarly, if there are n conditions then 2n test cases would be required.

* **Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.
**Steps:**
	1. Make the corresponding control flow graph
	2. Calculate the cyclomatic complexity
	3. Find the independent paths
	4. Design test cases corresponding to each independent path

**Flow graph notation:** It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.


**Cyclomatic Complexity:** It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity.
Calculating V(G):

* 1. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
	2. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
	3. V(G) = Number of non-overlapping regions in the graph

**Example:**

V(G) = 4 (Using any of the above formulae)
No of independent paths = 4

* 1. #P1: 1 – 2 – 4 – 7 – 8
	2. #P2: 1 – 2 – 3 – 5 – 7 – 8
	3. #P3: 1 – 2 – 3 – 6 – 7 – 8
	4. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8
* **Loop Testing:** Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.
	1. **Simple loops:** For simple loops of size n, test cases are designed that:
		+ Skip the loop entirely
		+ Only one pass through the loop
		+ 2 passes
		+ m passes, where m < n
		+ n-1 ans n+1 passes
	2. **Nested loops:** For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
	3. **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each.
	If they’re not independent, treat them like nesting.

**Advantages:**

1. White box testing is very thorough as the entire code and structures are tested.
2. It results in the optimization of code removing error and helps in removing extra lines of code.
3. It can start at an earlier stage as it doesn’t require any interface as in case of black box testing.
4. Easy to automate.

**Disadvantages:**

1. Main disadvantage is that it is very expensive.
2. Redesign of code and rewriting code needs test cases to be written again.
3. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.
4. Missing functionalities cannot be detected as the code that exists is tested.
5. Very complex and at times not realistic.