| **S. No.** | **Black Box Testing** | **White Box Testing** |
| --- | --- | --- |
| 1. | It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| 2. | Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| 3. | It is mostly done by software testers. | It is mostly done by software developers. |
| 4. | No knowledge of implementation is needed. | Knowledge of implementation is required. |
| 5. | It can be referred to as outer or external software testing. | It is the inner or the internal software testing. |
| 6. | It is a functional test of the software. | It is a structural test of the software. |
| 7. | This testing can be initiated based on the requirement specifications document. | This type of testing of software is started after a detail design document. |
| 8. | No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| 9. | It is the behavior testing of the software. | It is the logic testing of the software. |
| 10. | It is applicable to the higher levels of testing of software. | It is generally applicable to the lower levels of software testing. |
| 11. | It is also called closed testing. | It is also called as clear box testing. |
| 12. | It is least time consuming. | It is most time consuming. |
| 13. | Example: Search something on google by using keywords | Example: By input to check and verify loops |

**Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.



## Black Box Testing Techniques

Following are the prominent[Test Strategy](https://www.guru99.com/how-to-create-test-strategy-document.html)amongst the many used in Black box Testing

* **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
* **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.

# Boundary Value Analysis and Equivalence Partitioning Testing

We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all test scenarios are covered. We use two techniques – **Equivalence Partitioning & Boundary Value Analysis testing techniques** to achieve this.

# Boundary Value Analysis

Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

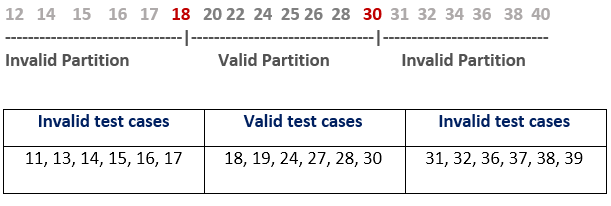
There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

Boundary Value Analysis

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

**Let's understand via practical:**

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.



The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

## What is Equivalence Class Partitioning?

*Equivalence class partitioning is a black-box testing technique or specification-based testing technique in which we group the input data into logical partitions called equivalence classes.*

*All the data items lying in an equivalence class are assumed to be processed in the same way by the software application to be tested when passed as input.*

So, instead of testing all the combinations of input test data, we can pick and pass any of the test data from a particular equivalence class to the application and assume that the application will behave in the same way for the other test data of that class. Let’s understand this with the help of an example.

## Example

Consider an example of an application that accepts a numeric number as input with a value between 10 to 100 and finds its square. Now, using equivalence class testing, we can create the following equivalence classes-

| **Equivalence Class** | **Explanation** |
| --- | --- |
| **Numbers 10 to 100** | This class will include test data for a positive scenario. |
| **Numbers** **0 to 9** | This class will include test data that is restricted by the application. Since it is designed to work with numbers 10 to 100 only. |
| **Greater than 100** | This class will again include test data that is restricted by the application but this time to test the upper limit. |
| **Negative numbers** | Since negative numbers can be treated in a different way so, we will create a different class for negative numbers in order to check the robustness of the application. |
| **Alphabets** | This class will be used to test the robustness of the application with non-numeric characters. |
| **Special characters** | Just like the equivalence class for alphabets, we can have a separate equivalence class for special characters. |

## Identification of Equivalence Classes

* Cover all test data types for positive and negative test scenarios. We have to create test data classes in such a way that covers **all sets of test scenarios** but at the same time, there **should not be any kind of redundancy**.
* If there is a possibility that the test data in a particular class can be treated differently then it is better to **split that equivalence class**.

## Advantages of Equivalence Classes Testing

* With the help of equivalence class testing, the number of test cases greatly reduces maintaining the same test coverage.
* This testing technique helps in delivering a quality product within a minimal time period.
* It is perfectly suitable for software projects with time and resource constraints.

## Disadvantages of Equivalence Classes Testing

* The whole success of equivalence class testing relies on the identification of equivalence classes. The identification of these classes relies on the ability of the testers who creates these classes and the test cases based on them.
* In the case of complex applications, it is very difficult to identify all set of equivalence classes and requires a great deal of expertise from the tester’s side.
* Incorrectly identified equivalence classes can lead to lesser test coverage and the possibility of defect leakage.

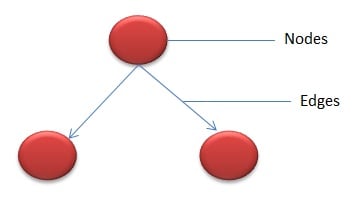
## What is McCabe’s Cyclomatic Complexity?

**Cyclomatic Complexity in Software Testing** is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program.

Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.

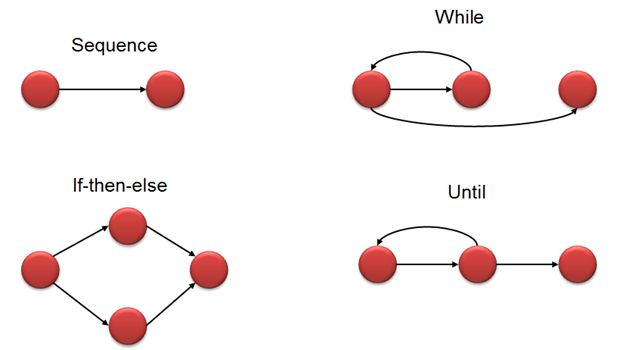
This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.

****

## Flow graph notation for a program:

Flow Graph notation for a program defines several nodes connected through the edges. Below are Flow diagrams for statements like if-else, While, until and normal sequence of flow.



## How to Calculate Cyclomatic Complexity

**Mathematical representation:**

Mathematically, it is set of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula –

V(G) = E - N + 2

**Where,**

E – Number of edges

N – Number of Nodes

V (G) = P + 1

Where P = Number of predicate nodes (node that contains condition)

**Example –**

i = 0;

n=4; //N-Number of nodes present in the graph

while (i<n-1) do

j = i + 1;

while (j<n) do

if A[i]<A[j] then

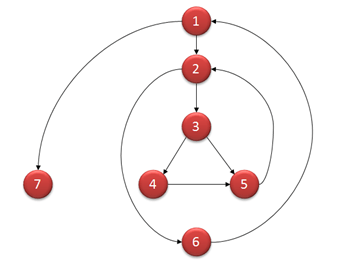
swap(A[i], A[j]);

end do;

j=j+1;

end do;

Flow graph for this program will be



**Computing mathematically,**

* V(G) = 9 – 7 + 2 = 4
* V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
* Basis Set – A set of possible execution path of a program
* 1, 7
* 1, 2, 6, 1, 7
* 1, 2, 3, 4, 5, 2, 6, 1, 7
* 1, 2, 3, 5, 2, 6, 1, 7
* ollowing table gives overview on the complexity number and corresponding meaning of v (G):

|  |  |
| --- | --- |
| **Complexity Number** | **Meaning** |
| 1-10 | Structured and well written code  High Testability  Cost and Effort is less |
| 10-20 | Complex Code  Medium Testability  Cost and effort is Medium |
| 20-40 | Very complex Code  Low Testability  Cost and Effort are high |
| >40 | Not at all testable  Very high Cost and Effort |

## Tools for Cyclomatic Complexity calculation:

Many tools are available for determining the complexity of the application. Some complexity calculation tools are used for specific technologies. Complexity can be found by the number of decision points in a program. The decision points are if, for, for-each, while, do, catch, case statements in a source code.

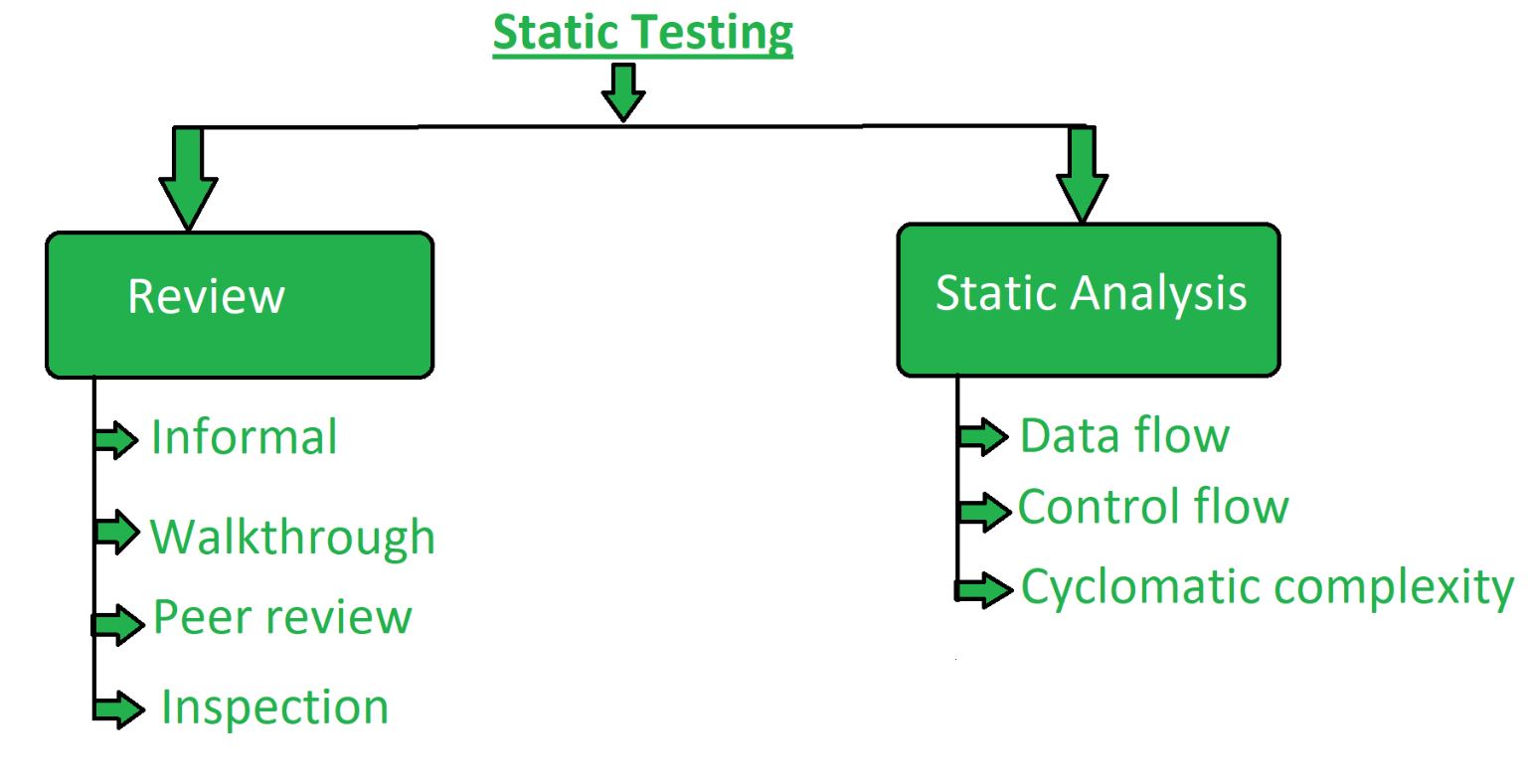
**Examples of tools are**

* [OCLint](https://github.com/oclint/oclint) – Static code analyzer for C and Related Languages
* Reflector Add In – Code metrics for .NET assemblies
* [GMetrics](https://github.com/dx42/gmetrics) – Find metrics in[Java](https://www.guru99.com/java-tutorial.html)related applications

## Uses of Cyclomatic Complexity:

Cyclomatic Complexity can prove to be very helpful in

* Helps developers and testers to determine independent path executions
* Developers can assure that all the paths have been tested atleast once
* Helps us to focus more on the uncovered paths
* Improve code coverage in Software Engineering
* Evaluate the risk associated with the application or program
* Using these metrics early in the cycle reduces more risk of the program

**Static Testing** is a type of a [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) method which is performed to check the defects in software without actually executing the code of the software application. Whereas in Dynamic Testing checks, the code is executed to detect the defects. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing. **Static Testing Techniques:** There are mainly two type techniques used in Static Testing: **Review:** In static testing review is a process or technique that is performed to find the potential defects in the design of the software. It is process to detect and remove errors and defects in the different supporting documents like software requirements specifications. People examine the documents and sorted out errors, redundancies and ambiguities. Review is of four types:

* **Informal:** In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.
* **Walkthrough:** It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.
* **Peer review:** Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.
* **Inspection:** Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

2. **Static Analysis:** Static Analysis includes the evaluation of the code quality that is written by developers. Different tools are used to do the analysis of the code and comparison of the same with the standard. It also helps in following identification of following defects:

**(a)** Unused variables

**(b)** Dead code

**(c)** Infinite loops

**(d)** Variable with undefined value

**(e)** Wrong syntax

Static Analysis is of three types:

* **Data Flow:** Data flow is related to the stream processing.
* **Control Flow:** Control flow is basically how the statements or instructions are executed.
* **Cyclomatic Complexity:** Cyclomatic complexity defines the number of independent paths in the control flow graph made from the code or flowchart so that minimum number of test cases can be designed for each independent path.

## What is State Transition Testing?

**State Transition Testing** is a black box testing technique in which changes made in input conditions cause state changes or output changes in the Application under Test(AUT). State transition testing helps to analyze behaviour of an application for different input conditions. Testers can provide positive and negative input test values and record the system behavior.

It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

**State Transition Testing Technique**is helpful where you need to **test different system transitions.**

## When to Use State Transition?

* This can be used when a tester is testing the application for a finite set of input values.
* When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behavior for a sequence of input values.
* When the system under test has a dependency on the events/values in the past.

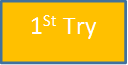
## When to Not Rely On State Transition?

* When the testing is not done for sequential input combinations.
* If the testing is to be done for different functionalities like exploratory testing

## Four Parts Of State Transition Diagram

There are 4 main components of the State Transition Model as below

**1) States** that the software might get



**2) Transition** from one state to another

https://www.guru99.com/images/1/103017_0527_WhatIsState2.png

**3) Events** that origin a transition like closing a file or withdrawing money

https://www.guru99.com/images/1/103017_0527_WhatIsState3.png

**4) Actions** that result from a transition (an error message or being given the cash.)



## State Transition Diagram and State Transition Table

There are two main ways to represent or design state transition, State transition diagram, and state transition table.

In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.

In state transition table all the states are listed on the left side, and the events are described on the top. Each cell in the table represents the state of the system after the event has occurred. It is also called State Table. It is useful in identifying invalid transitions.

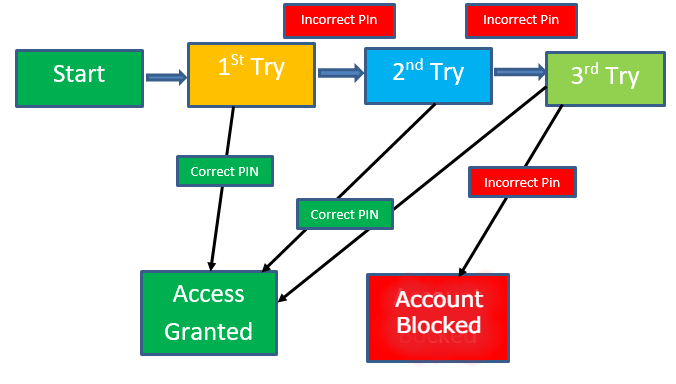
## How to Make a State Transition (Examples of a State Transition)

### Example 1:

Let’s consider an ATM system function where if the user enters the invalid password three times the account will be locked.

In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.

### State transition diagram



In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

### State Transition Table

|  |  |  |
| --- | --- | --- |
|  | **Correct PIN** | **Incorrect PIN** |
| **S1) Start** | **S5** | **S2** |
| **S2) 1st attempt** | **S5** | **S3** |
| **S3) 2nd attempt** | **S5** | **S4** |
| **S4) 3rd attempt** | **S5** | **S6** |
| **S5) Access Granted** | **–** | **–** |
| **S6) Account blocked** | **–** | **–** |

In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

### White Box Testing Tools

Given below is a list of top white box test tools.

**#1) Veracode**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2015/02/Veracode.jpg)

Veracode’s white box testing tools will help you in identifying and resolving the software flaws quickly and easily at a reduced cost. It supports several application languages like .NET, C++, JAVA etc and also enables you to test the security of desktop, web as well as mobile applications. Still, there are several other benefits of Veracode tool. For detailed information about Veracode White box test tools, please check the below link.

Website Link: [Veracode](https://www.veracode.com/security/white-box-testing-tools)

**#2) EclEmma**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2015/02/EclEmma.jpg)

EclEmma was initially designed for test runs and analysis within the Eclipse workbench. It is considered to be a free Java code coverage tool and has several features as well. To install or know more about EclEmma please check out the below link.

Website Link: [EclEmma](http://www.eclemma.org/download.html)

**#3)RCUNIT**

[RCUnit](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2015/02/RCUnit.jpg)

A framework which is used for testing C programs is known as RCUNIT. RCUNIT can be used accordingly based on the terms of the MIT License. It is free to use and in order to install or know more about it, please check the below link.

Website Link: [RCUNIT](http://sourceforge.net/projects/rcunit/)

**#4) cfix**

cfix is one of the unit testing frameworks for C/C++ which solely aims at making test suites development as simple and easy as possible. Meanwhile, cfix is typically specialized for NT Kernel mode and Win32. To install and know more about cfix, please check out the below link

Website Link: [cfix](http://sourceforge.net/projects/cfix/files/latest/download)

**#5) Googletest**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2015/02/Googletest.jpg)

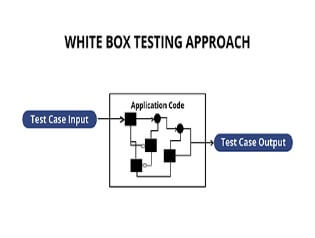
Googletest is Google’s C++ test framework. Test Discovery, Death tests, Value-parameterized tests, fatal & non-fatal failures, XML test report generation etc are few features of GoogleTest but there are several other features too. Linux, Windows, Symbian, Mac OS X are few platforms where GoogleTest has been used. In order to Download, please check the below link.

Download Link: [Googletest](https://github.com/google/googletest/archive/master.zip)

## **White Box Testing**

A chunk of code is run during a white-box test case with predetermined input values to confirm predetermined output values. A bug is discovered if these values do not match. This is stated numerous times throughout the application. **White-box testing aims to confirm:**

* Expected results
* Verify specific code portions and make sure all testing has been done.
* Conditional statements, incomplete pathways



In white box penetration testing, the tester/developer is fully aware of the application's source code, comprehensive network information, all relevant IP addresses, and all server information the application uses. In order to uncover security risks, it is intended to attack the code from many sides. White box penetration testing entails looking for security flaws in the internal code, enhancing security by making it more aesthetically pleasing and functional, and fixing poorly organised development routes. It includes comparing numerous defined inputs with anticipated outputs. In order to find security flaws or hazards as quickly as possible, this technique is helpful.

The most crucial component of the white box testing approach is the code coverage analysis, which enables a team of software engineers to identify areas of code that a particular set of test cases doesn't really run, hence enhancing the quality of a software programme. The analysis of code coverage can be done using a variety of methodologies. **These include:**

1. Coverage of Statements: This method is employed to test each potential assertion at least once. When employing this method, Cantata++ is the preferred tool.
2. Decision-making coverage: This entails running each potential conditional loop and decision condition at least once. The recommended tool for this technique is TCAT-PATH, which supports C, C++, and Java applications.
3. Condition Coverage: When all the conditions have been tested, this makes one code execution necessary.
4. Decision/Condition Coverage: This combined technique is used to test every decision and condition coverage at least once as the code is running.
5. Multiple Condition Coverage: In this style of white box testing, each system entry point must be tested at least once.

**The various advantages and disadvantages of White Box Testing that are listed below:**

## **Advantages of White Box Testing**

The completeness, automation, time savings, optimization, and introspection of white box testing are its benefits.

**1. Thoroughness**

Complete code coverage is the fundamental tenet of white-box testing. The basic concept is to test as much code as you can, which is far more thorough than standard black-box testing. White-box testing's thoroughness also lends it a distinct framework. The rules of testing must be precise, engineering-based, and well-defined. This type of testing is transparent, it is possible to do thorough tests that cover all possible paths as well as the complete structure and code base. It evaluates internal and external vulnerabilities as well, which could aid in preventing future security threats and assaults.

**2. Unit Testing**

Unit testing is made possible by understanding the application's internal workings. As the name implies, unit tests examine single lines of code, or units, to determine whether they function as intended. These tests are easy to perform programmatically, allowing developers to rapidly determine whether something is broken. Unit tests are a useful tool for determining whether a previously functioning component has recently broken.

**3. Time**

Time management is a top responsibility during the software development process because there are constant deadlines to satisfy. White-box testing has the ability to drastically speed up the testing process. Developers frequently have a general understanding of the problem and the best way to resolve it as soon as they discover a fault. The expense of communication between developers and QA is also eliminated by white-box testing because developers can identify and address problems without waiting for QA.

**4. Optimization**

A section-by-section analysis of the code enables developers to eliminate unnecessary code or condense already-existing code. Additionally, by removing obfuscated problems that can go undetected during routine testing, code can be made more efficient.

**5. Introspection**

White-box testing enables programmers to thoroughly consider implementation. Developers are compelled to think about the relationships between different pieces of code. Perhaps the existing implementation is adequate but will not scale well in the future or contains extraneous components that can be removed. Developers can review designs and consider how they might be improved by using white-box testing.

## **Disadvantages of White Box Testing**

White-box testing has drawbacks including high cost, frequently changing code, and missing cases.

**1. Expensive**

White-box testing becomes highly time and money-consuming to undertake as it is more thorough. Although this is somewhat mitigated by unit testing, writing the unit tests requires an initial investment. Additionally, this kind of testing may not scale well with huge applications. Testing every code version becomes very difficult. White-box testing necessitates competent testers who are familiar with programming, in contrast to black-box testing. This drives up the cost and may discourage developers from working on additional features. White-box testing must take all of these costs into account.

**2. Code base that changes quickly**

If the code base is changing quickly, automated test cases are useless. Most written test cases are frequently useless after redesigns or rework and require rewriting. If the implementation changes frequently, an updated test script is necessary.

**3. Incomplete cases**

Only existing functionalities are validated and tested during white-box testing. White-box testing won't detect a feature that is only partially implemented or that has certain components missing. Black-box testing that is driven by requirements excels in this area.

**4. Time consuming**

When employing the white box testing approach for large applications, exhaustive testing becomes even more difficult. White box testing takes a lot of time because it requires creating a wide range of inputs to test every possible path and circumstance.

**5. More errors**

It is not realistic to test every condition, thus some might go untested. With the general method of analysing each line by line or path by path, errors in the code might not be found or might even be introduced.