ISSN: 2321-7782 (Online)
Volume 3, Issue 10, October 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

# A Comparative Study of Black Box Testing and White Box Testing Techniques

**Manish Kumar**[1]
Assistant Professor,
University Department of Computer Applications,
Vinoba Bhave University, Hazaribag, Jharkhand, India

**Santosh Kumar Singh**[2]
Assistant Professor,
University Department of Computer Applications,
Vinoba Bhave University, Hazaribag, Jharkhand, India

**Dr. R. K. Dwivedi**[3]
Associate Professor,
University Department of Mathematics,
Vinoba Bhave University, Hazaribag, Jharkhand, India

***Abstract:*** ***Software testing is the most important and time consuming part of software development life cycle It's purpose is to detect software failures so that defects may be recovered and corrected in early phase. Software Testing is a process of confirming that the product/software that has been manufactured by programmers is a quality product and to assure that the manufactured product is working according to the specification and satisfying the customer needs. There are many approaches to software testing, but effective testing of complex product is essentially a process of investigation, not merely a matter of creating and following route procedure. It is not possible to find out all the errors in the program. This fundamental problem in testing thus throws an open question, as to what would be the strategy we should adopt for testing. In this paper, we have described and compared the two most prevalent and commonly used software testing techniques for detecting errors, they are: Black Box Testing and White Box Testing.***

***Keywords: Software Testing, Black Box Testing, White Box Testing, Software Developnment Life Cycle, Software Quality***

## I. INTRODUCTION

Software testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong, it's human tendency.

The primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing can be used as a generic metric as well. Software testing is also used to test the software for other software quality factors like reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility etc. [1][2]

Software testing is a very broad area, which involves many technical and non-technical areas, such as specification, design and implementation, maintenance, process and management issues in software engineering. Our study focuses on the state of the art in testing techniques, as well as the latest techniques which representing the future direction of this area. Before stepping into any further detail let's look into the two important testing techniques.

The two most important techniques that are used for finding errors are Black Box Testing and White Box Testing. Black Box Testing is a testing technique without reference to the internal structure of the component or system. In Black Box Testing it is not necessary for a tester to have good programming knowledge, since it only examines the fundamental aspects of the system without going into detail. [2]. White Box Testing is a testing technique based on the internal structure of the component or system. In White Box Testing it is necessary for a tester to have good programming knowledge, so to better understand the source code. White Box Testing can be performed any time in the life cycle after the code is developed. [2]

In this paper our main aim is to discover when to go for black box testing and when to go for white box testing. The remainder of this paper discusses types, methods, tools and techniques used for black box and white box testing. We have also discuss here the comparison between the two important testing techniques.

This paper is organized as follows; section 2 presents different Black Box Testing Technique, Section 3 presents different White Box Testing Technique, Section 4 presents Comparison between Black box testing and White box testing, and Section 5 presents the conclusion that we drew.

## II. BLACK BOX TESTING TECHNIQUE

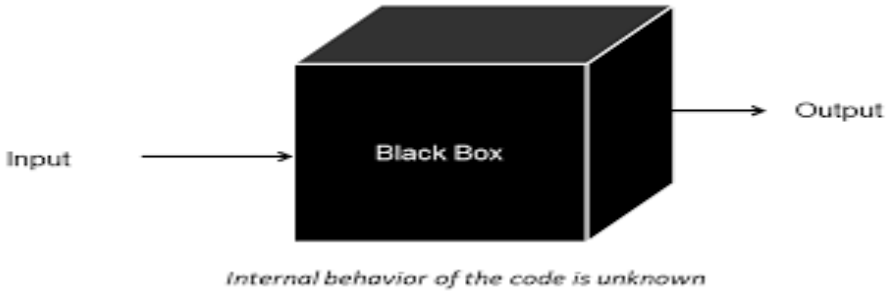



***Fig1 : Represent Black Box Testing***

Black Box Testing is testing without knowledge of the internal working of the application under test (AUT). Also known as functional testing or input output driven testing. A software testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on AUT the tester only knows the inputs and what the expected outcomes should be and how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications. For this reason, the tester and the programmer can be independent of one another, avoiding programmer's biasness towards his own work. This method of test design is applicable to all levels of software testing: unit, integration, functional testing, system and acceptance. [3]

Black Box Testing is also known as clear box testing, glass box testing, transparent box testing, and structural testing.

*a) Types Of Black Box Testing [4][5][6]*

1. Build Verification Testing (BVT)
2. Smoke Testing
3. Sanity Testing
4. User Interface Testing
5. Usability Testing
6. Integration Testing
7. Compatibility Testing
8. Retesting
9. Regression Testing
10. Performance Testing
11. Load Testing
12. Stress Testing
13. Volume Testing

14. System Testing

15. Acceptance Testing

***1. Build Verification Test (BVT)***

A build verification test is a set of tests run on each new build of a product to verify that the build is send to the test team. The Build Verification Test is generally a set of tests, which exercises the mainstream functionality of the application. Any build that fails the build verification test is rejected, and testing continues on the previous build (provided there has been at least one build that has passed the verification test). So build verification test are a type of regression that is done every time a new build is conceived. Build Verification tests are important because they led developers know right away if there is a serious problem with the build, and they save the test team wasted time and frustration testing that is done every time a new build is taken.

***2. Smoke Testing***

Smoke testing is done by developers before the build is released or by testers before accepting a build for further testing. After code reviews, smoke testing is the most cost effective method for identifying and fixing defects in software. In software engineering, a smoke test generally consists of a collection of tests that can be applied to a newly created or repaired computer program. This is a "shallow and wide" approach to the application. The tester "touches" all areas of the application without getting too deep, looking for answers to basic questions like, " Can I launch the test item at all?", "Do the buttons on the window do things?". The purpose is to determine whether or not the application is so badly broken that testing functionality in a more detailed way is unnecessary. These written tests can either be performed by the same process that generates the build itself. This is sometimes referred to as "rattle" testing-as in " if I shake it does it rattle?"

***3. Sanity Testing***

Sanity testing is a quick, broad and shallow testing performed whenever a cursory testing is sufficient to prove the application is functioning according to specifications. If the smoke test fails, it is imposible to conduct a sanity test. The ideal sanity test exercises the smallest subset of application functions needed to determine whether the application logic is generally functional and correct (for example, an interest rate calculation for a financial application). If the sanity test fails, it is not reasonable to attempt more rigorous testing. Both sanity tests and smoke tests are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing.

***4. User Interface Testing***

User-interface testing is the testing of the user interface to ensure that it follows accepted standards and meets it requirements. User-interface testing is often referred to as graphical user interface (GUI) testing or UI testing i.e. testing the interface extensions of the application to the user. This includes how the application handles keyboard and mouse input and how it displays screen text, images, buttons, menus, dialog boxes, icons, toolbars and more.

***5. Usability Testing***

Usability testing is a black-box testing technique. The aim is to observe people using the product to discover errors. Usabilty testing generally involves measuring how well respond in four areas: efficiency, accuracy, recall, and emotional response.

Performance-How much time, and how many steps, are required for people to complete basic tasks? (For example, find something to buy, create a new account, and order the item.)

Accuracy- How many mistakes did people make? (And were they fatal or recoverable with the right information?)

Recall- How much does the person remember afterwards or after periods of non-use?

Emotional response- How does the person feel about the task completed ? Is the person confident, stressed? Would the user recommend this system to a friend?

***6. Integration Testing***

One of the most difficult aspects of software development is the integration and testing of large, untested sub-systems. The integrated system frequently fails in significant and mysterious ways, and it is difficult to fix it. Integration testing exercises several units that have been combined to form a module, subsystem, or system. Integration testing focuses on the interfaces between units, to make sure the units work together.

There are three main approaches to integration testing: top-down, bottom-up and 'big bang'.

Top-down combines, tests, and debugs top-level routines that become the test 'harness' or 'scaffolding' for lower-level units. Bottom-up combines and tests low-level units into progressively larger modules and substems. 'Big bang' testing is, unfortunately, the prevalent integration test 'method'. This is waiting for all the module units to be complete before trying them out together.

***7. Compatibility Testing***

Compatibilty testing, part of software is testing conducted on the application to evaluate the application's with the computing environment. Computing environment may contain some or all of the below mentioned elements:

Database (Oracle, Sybase, DB2, etc.)

Other System Software (Web server, networking/messaging tool, etc.)

Browser Compatibility (Firefox, Netscape, Internet Explorer, Safari, etc.)

***8. Retesting***

Retesting is a type of testing in which the tester checks that the defects reported in the previous build have been fixed. This requires re-running the failed test cases.

Consider an example in which 10 defects have been reported in build 1.2. These 10 defects will be assigned to developers for fixing. When all the ten defects have been fixed, the new build with the changes will be retested to verify that the defects have really been fixed. This is retesting.

***9. Regression Testing***

Regression testing is a type of software testing in which we check for new bugs introduced due to fixing the reported bugs or changes made in the previous build.

Here we also check that the bugs reported in the earlier build have been fixed i.e. we re-test. This requires re-running the failed test cases. Regression testing is often accomplished through the construction, execution and analysis of product and system tests. Regression testing is an expensive but necessary activity performed on modified software to provide confidence that changes are correct and do not adversely affect other system components.

***10. Performance Testing***

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort. It deals with testing how well an application compiles to performance requirements.

Performance testing can serve different purposes. It can demonstrate that the system meets performance criteria. It can compare two systems to find which performs better. Or it can measure what parts of the system or workload cause the system to perform badly. In the diagnostic case, software engineer use tools such as profilers to measure what parts of a device or software contribute most to the poor performance or to establish throughput levels for maintained acceptable response time. It is critical to the cost performance of a new system that performance test efforts begin at the inception of the development project and extend through to deployment. The later a performance defect is detected, the higher the cost of remediation. This is true in case of functional testing, but even more so with performance testing, due to the end-to-end nature of its scope.

***11. Load Testing***

This is the simplest form of performance testing. A load test is usually conducted to understand the behavior of the application under for a specific expected load. This load can be the expected number of concurrent users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc are also monitored, then this simple test can itself point toward the bottleneck in the application.

***12. Stress Testing***

Stress testing is the testing conducted to evaluate a system or component at or beyond the limits of its specified requirement (requirement of load handling or bearing). The main purpose is to ensure that the system fails and recovers gracefully. It is done by locating a point, by either gradually increasing the load or depleting the systems resources, where the applications cresses or fails to respond. The objective is to test for scalability and recoverability of the applications under test

***13. Volume Testing***

Volume testing tests an application for a certain data volume. This volume can be, in generic terms, the database size or it could also be the size of an interface file that is the subject of volume testing. for example, if you want to volume test your applications with a specific database size, you will explode you data base to that size and then test the applications performance on it.

***14. System Testing***

System testing is conducted on a complete, integrated system to evaluate the systems compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware systems. System testing is a more limiting type of testing: it seeks to detect defects both within the "inter – assemblages" and also within the system as a whole.

***15. Acceptance Testing***

Acceptance testing is a high level testing procedure which insure that an application behaves as accepted by the client. Acceptance test operate on a fully integrated application. They are done to ensure that the application has no error of the acceptance criterion; established for the product. Acceptance testing is classified in two categories:

a) Alpha testing: alpha testing is simulated or actual operational testing by potential users/ customer or an independent test team at the developer site. Alpha testing is often employed for off-the –shelf software as a form of internal acceptance testing, your alpha test will be a test among the vendors teams to confirm that the product operational as per the established acceptance norms

b) Beta testing: beta testing comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside the vendor team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of futures users. In software development, a beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. Beta testing can be consider "pre – release testing"

***Acceptance testing is important:***

1. It covers the acceptance criterion of the client/ end user

2. It measures when functionality valued by the client is complete from acceptance criterion perspectives

*B. Methods of Black box Testing*

1. Equivalence Class Partitioning

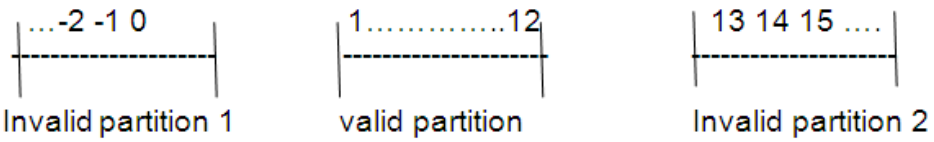
2. Boundary Value Analysis

3. Decision Tables and State Transition Diagram

1. Equivalence Class Partitioning [6]

Equivalence partitioning (EP) is a specification-based or black-box technique. It can be applied at any level of testing and is often a good technique to use first. The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. Equivalence partitions are also known as equivalence classes – the two terms mean exactly the same thing. In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

This may be best explained by the example which takes a parameter "month". The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition. In this example there are two further partitions of invalid ranges. The first invalid partition would be <=0 and the second invalid partition would be >=13




The testing theory related to equivalence partitioning says that only one test case of each partition is needed to evaluate the behavior of the program for the related partition. In other words it is sufficient to select one test case out of each partition to check the behavior of the program. To use more or even all test cases of a partition will not find new faults in the program. The values within one partition are considered to be "equivalent". Thus the number of test cases can be reduced considerably.

Equivalence partitioning is no stand alone method to determine test cases. It has to be supplemented by boundary value analysis. Having determined the partitions of possible inputs the method of boundary value analysis has to be applied to select the most effective test cases out of these partitions.

2. Boundary Value Analysis [6][7]

Boundary value analysis (BVA) is based on testing at the boundaries between partitions. Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions). Boundary value analysis is the technique of making sure that behavior of system is predictable for the input and output boundary conditions.Reason why boundary conditions are very important for testing is because defects could be introduced at the boundaries very easily.

For example, if you were to write code to simulate following condition- "input should be greater than equal to 10 and less than 50 " Probably you will write something like

If(input>=10 AND input<50) then

Do some

Else

Do something else

So, according to this input values from 10 to 49 are valid , but if you make mistake in specifying the conditions, following things can happen

Input>10 AND input<50-----> input value 10 in invalid now.

Input<=10 AND input <50-----> input value 9 is valid now.

Input>=10 AND input <50-----> input value 50 is valid now.

Input>=10 AND input >50-----> input value 49 is valid now.

Because it is very easy to introduce defects at boundaries, boundary values are important. So for the above example, at the minimum we should have following test cases for boundaries 9, 10, 11 and 48, 49, 50 lower_boundary – 1, lower_boundary + 1 and upper_boundary, upper_boundary+1

3. Decision Tables and State Transition Diagrams [6][7]

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface. The other two specification-based software testing techniques, decision tables and state transition testing are more focused on business logic or business rules.

A decision table is a good way to deal with combinations of things (e.g. inputs).This technique is sometimes also referred to as a 'cause-effect' table. The reason for this is that there is an associated logic diagramming technique called 'cause-effect graphing' which was sometimes used to help derive the decision table (Myers describes this as a combinatorial logic network. Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers. Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules. It helps the developers to do a better job can also lead to better relationships with them. Testing combinations can be a challenge, as the number of combinations can often be huge. Testing all combinations may be impractical if not impossible. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

A State transition diagram is used where some aspect of the system can be described in what is called a 'finite state machine'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to

another are determined by the rules of the 'machine'. This 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. A finite state system is often shown as a state diagram.One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be. Where a part of the system is more important (that is, requires more testing) a greater depth of detail can be modeled. Where the system is less important (requires less testing), the model can use a single state to signify what would otherwise be a series of different states.

A state transition model has four basic parts:

a) The states that the software may occupy (open/closed or funded/insufficient funds);

b) The transitions from one state to another (not all transitions are allowed);

c) The events that cause a transition (closing a file or withdrawing money);

d) The actions that result from a transition (an error message or being given your cash).

Hence we can see that in any given state, one event can cause only one action, but that the same event – from a different state – may cause a different action and a different end state.

For example, if you request to withdraw $100 from a bank ATM, you may be given cash. Later you may make exactly the same request but it may refuse to give you the money because of your insufficient balance. This later refusal is because the state of your bank account has changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal. A state diagram can represent a model from the point of view of the system, the account or the customer.

*C. Tools used for Black Box Testing [8]*

Black box testing tools are mainly record and playback tools. These tools are used for regression testing that to check whether new build has created any bug in previous working application functionality. These record and playback tools records test cases in the form of some scripts like TSL, VB script, Java script, Perl.

The following are some black box testing tools collected:

Typhon - http://www.ngssoftware.com/products/internet-security/ngs-typhon.php9

MatriXay - http://www.dbappsecurity.com7

NGSSQuirreL - http://www.ngssoftware.com/products/database-security/2

Watchfire AppScan - http://www.watchfire.com2

*D. Advantages and Disadvantages of Black Box Testing technique are listed below [9]*

Advantages of black box testing:

1. Black box tests are reproducible.

2. The environment the program is running is also tested.

3. The invested effort can be used multiple times.

4. Tests are done from a user's point of view.

5. Efficient when used on a larger system.

Disadvantages of black box testing:

1. The results are often overestimated.

2. Not all properties of a software product can be tested.

3. The reason for a failure is not found.

4. May leave many program path untested.

5. Test cases are tough and even challenging to design.

## III. WHITE BOX TESTING TECHNIQUE

This paper proposes the IND-OCPA-P model to analyze the security of the proposed EOB and the encryption schemes supporting an efficient range query over encrypted data.

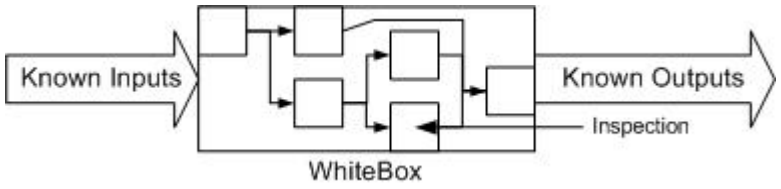



***Fig2: Represent White Box Testing***

White Box Testing is performed on the knowledge of how the system is implemented. White Box Testing includes analyzing data flow, control flow, information flow, coding practices, and exception and error handling within the system, to test the intended and unintended software behavior. White Box Testing can be performed to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities. White Box Testing requires access to the source code. Though White Box Testing can be performed any time in the life cycle after the code is developed, it is a good practice to perform White Box Testing during the unit testing phase. [9]

White Box Testing is also known as clear, open, structural, and glass box testing.

*A. Types Of White Box Testing. [9]*

1. Control Flow Testing

2. Statement and Branch Coverage Testing

3. Path Testing

4. Data Flow Testing

5. Loop Testing

***1. Control Flow testing***

It is a structural testing strategy that uses the program control flow as a model control flow and favours more but simpler paths over fewer but complicated path. Control-flow behavioural testing applies to almost all software and is effective for most software. It is a fundamental technique. Its applicability is mostly to relatively small programs or segments of larger programs.

Fundamental Path Selection Criteria:

» Ensure every instruction in the routine has been exercised at least once

» Every decision (branch or case statement) has been taken in each possible direction at least once

» A sufficient number of paths to achieve coverage

» Selection of short, functionally sensible paths

» Minimizing the number of changes from path to path - preferably only one decision changing at a time

***2. Statement coverage & Branch coverage***

In programming language, statement is nothing but the line of code or instruction for the computer to understand and act accordingly. A statement becomes an executable statement when it gets compiled and converted into the object code and performs the action when the program is in running mode.

Hence "Statement Coverage", as the name suggests, is the method of validating that each and every line of code is executed at least once."Branch" in programming language is like the "IF statements". If statement has two branches: true and false.So in Branch coverage (also called Decision coverage), we validate that each branch is executed at least once.In case of a "IF statement", there will be two test conditions:

(i) One to validate the true branch and

(ii) (ii) Other to validate the false branch

Hence in theory, Branch Coverage is a testing method which when executed ensures that each branch from each decision point is executed.

***3. Path Testing***

Path coverage tests all the paths of the program. This is a comprehensive technique which ensures that all the paths of the program are traversed at least once. Path Coverage is even more powerful that Branch coverage. This technique is useful for testing the complex programs.

Taking up each and every individual path through which the flow of code taken place.

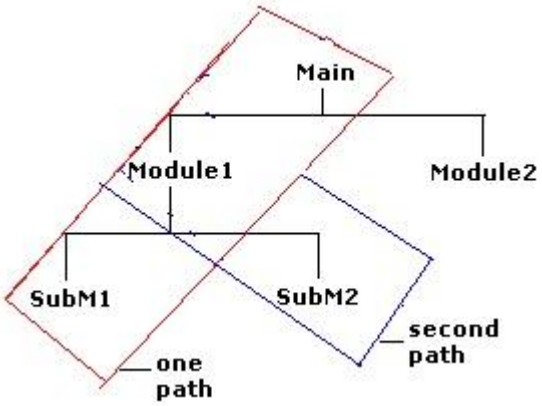



***Fig 3: Path Testing***

***4. Data Flow Testing***

A white box test design technique in which test cases are designed to execute definition and use pairs of variables. In this type of testing the control flow graph is annoted with the information about how the program variables are define and used. Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects. Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

Data Flow testing helps us to pinpoint any of the following issues:

a) A variable that is declared but never used within the program.

b) A variable that is used but never declared.

c) A variable that is defined multiple times before it is used.

d) Deallocating a variable before it is used.

***5. LoopTesting***

Loop testing is a White Box Testing techniques that focuses exclusively on validity of loop construct. A Piece of code executing continuously until the condition becomes false and testing whether it is proper or not.

What is tested in Loop Testing?

a) Loops Testing reveals loops initialization problems.

b) By going through the loop once, the uninitialized variables in the loop can be determined.

c) Testing can also fix loop repetition issues.

d) Loops can also reveal capacity/performance bottlenecks.

Ex: For(Loop=1;Loop<=10;Loop++)

{

-------

--------

}

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

White box testing, is divided it into two basic steps**:**

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.

*C. Why and When White-Box Testing:*

In white box testing we not only check the way things are working but also it supplements the black box testing as we can not make exhaustive test cases due to our limited resources and the cost-benefit trade-off. There are funtions/methods that would give you correct output after giving it inputs but there may be an input test case which you did not think about and this may create problem at the client side. So if you get a correct output for certian inputs, this doesn't mean that you have completly checked your function/method. For this we need white box testing to completly understand the mechanism and the way it is producing the output using certain input.

White box testing is done at low level design and implementable code. White box testing can be applied at the unit, Integration, and system levels of software testing process. Although traditional testers tended to think of white box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system- level test.

White box testing can be used for other development facts like requirements analysis, designing and test cases.

White box testing is mainly used for detecting logical errors in the program code. It is used for debugging a code, finding random typographical errors, and uncovering incorrect programming assumptions. 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

*D. Advantages and Disadvantages of White Box Testing technique are listed below [9]*

***Advantages of White Box Testing :***

1. To start the testing of the software no need to wait for the GUI, you can start the White Box Testing.
2. As covering all possible paths of code so this is a thorough testing.
3. Tester can ask about implementation of each section, so it might be possible to remove unused lines of code which might be causing introduction of bug.
4. As the tester is aware of internal coding structure, then it is helpful to derive which type of input data is needed to testing software application effectively.
5. White Box Testing allows you to help in the code optimization
6. White box testing give clear, engineering-based, rules for when to stop testing.

***Disadvantages of White Box Testing:***

1. To test the software application a highly skilled resource is required to carry out testing who know the deep knowledge of internal structure of the code which increase the cost.
2. If the application under test is large in size then exhaustive testing is impossible.
3. It is not possible for testing each and every path/condition of software program, which might miss the defects in code.
4. White Box Testing very expensive type of testing.
5. To analyze each line by line or path by path is nearly impossible work which may introduce or miss the defects in the code.

## IV. Comparison Between Black Box Testing And White Box Testing. [9][10]

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 Testing, System Testing | Mainly applicable to lower levels of testing: Unit Testing, Integration 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 |

*Fig: Table 1*

## V. CONCLUSION

Testing has been widely used as a way to help engineers develop high-quality systems, and the techniques for testing have evolved from an ad hoc activities means of small group of programmers to an organized discipline in software engineering. However, the maturation of testing techniques has been fruitful, but not adequate. Pressure to produce higher-quality software at lower cost is increasing and existing techniques used in practice are not sufficient for this purpose. To carry out software testing in a more effective manner, our paper describes and compare the two most important software testing techniques. Our study also focuses on the state of the art in testing techniques, as well as the latest techniques which will represent the future direction of this area.

Today, testing is the most challenging and dominating activity used by industry, therefore, improvement in its effectiveness, both with respect to the time and resources, is taken as a major factor by many researchers The purpose of testing can be quality assurance, verification, and validation or reliability estimation. It is a tradeoff between budget, time and quality. Software Quality is the central concern of software engineering. Testing is the single most widely used approach to ensuring software quality.

## References

1. Mohd. Ehmer Khan, "Different Forms of Software Testing Techniques for Finding Errors," IJCSI, Vol. 7, Issue 3, No 1, pp 11-16, May 2010
2. SoftwareTestingMethodologiesbyAzharavailableathttp://azhar-aperpresentation.blogspot.com/2010/04/software-testing-methodologies.html
3. Mohd. Ehmer Khan, "Different Approaches to Black Box Testing Technique for Finding Errors," IJSEA, Vol. 2, No. 4, pp 31-40, October 2011
4. Shivkumar Hasmukhrai Trivedi, "Software Testing Techniques", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 10, October 2012, ISSN: 2277 128X
5. Anitha.A, "A Brief Overview of Software Testing Techniques and Metrics", *International Journal of Advanced Research in Computer and Communication Engineering,Vol. 2, Issue 12, December 2013,* ISSN (Online) : 2278-102
6. http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf
7. Vineta Arnicane," Complexity of Equivalence Class and Boundary Value Testing Methods", Scientific Papers, University of Latvia, 2009. Vol. 751 Computer Science and Information Technologies 80–101 P.
8. http://community.sitepoint.com/t/black-box-testing-tools/5515
9. Mohd. Ehmer Khan, Farmeena Khan," A Comparative Study of White Box, Black Box and Grey Box Testing Techniques", *(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No.6, 2012*
10. http://www.cs.unh.edu/~it666/reading_list/Defense/blackbox_vs_whitebox_testing.pdf