Available Online at www.ijecse.org ISSN- 2277-1956

Bridge between Black Box and White Box – Gray Box Testing Technique

Shivani Acharya
Lecturer, Sir P.P Institute of Science
Email: - shivani.acharya@gmail.com

Vidhi Pandya
Lecturer, Msc. I.T Department
Email: - vidhibp@yahoo.com

Abstract- Software testing is a highly complex and time consuming activity- It is even difficult to say when testing is complete. The effective combination of black box (external) and white box (internal) testing is known as Gray-box testing. Gray box testing is a powerful idea if one knows something about how the product works on the inside; one can test it better, even from the outside. Gray box testing is not black box testing, because the tester does know some of the internal workings of the software under test. It is not to be confused with white box testing, testing approach that attempts to cover the internals of the product in detail. Gray box testing is a test strategy based partly on internals. This paper will present all the three methodology Black-box, White-box, Gray-box and how this method has been applied to validate critical software systems.

Keywords- Black-box, White-box, Gray-box or Grey-box

Introduction

In most software projects, testing is not given the necessary attention. Statistics reveal that the nearly 30-40% of the effort goes into testing irrespective of the type of project; hardly any time is allocated for testing. The computer industry is changing at a very rapid pace. In order to keep pace with a rapidly changing computer industry, software test must develop methods to verify and validate software for all aspects of the product lifecycle. Test case design techniques can be broadly split into two main categories: Black box & White box.

Black box + White box = Gray Box

Spelling:

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

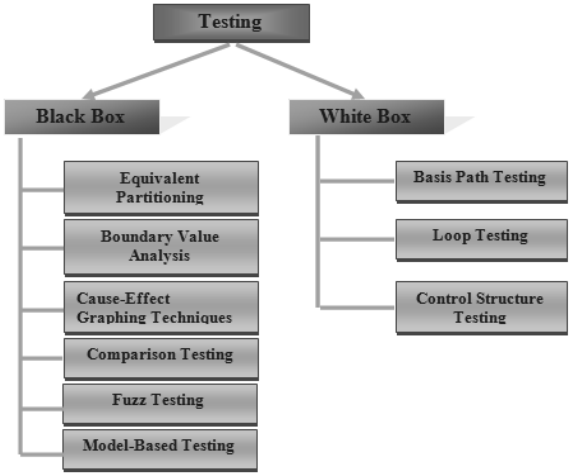
Gray Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term the more you know the better carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in Gray box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

The **gray-box testing** goes mainly with the testing of web applications because it considers high-level development, operating environment, and compatibility conditions. During black-box or white-box analysis it is harder to identify problems, related to end-to-end data flow. Context-specific problems, associated with web site testing are usually found during **gray-box verifying**.

IJECSE
International Journal of Electronics and Computer Science Engineering
WWW.IJECSE.ORG

Bridge between Black Box and White Box – Gray Box Testing Technique



Testing Methods

Fig 1: Classification of Test Techniques

1. Black Box Testing

Black box testing is a software testing techniques in which **functionality of the software under test (SUT) is tested without looking at the internal code structure**, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

Black box testing is best suited for rapid test scenario testing and quick Web Service prototyping. This testing technique for Web Services provides quick feedback on the functional readiness of operations through quick spot checking. Black Box testing is also better suited for operations that have enumerated inputs or tightly defined ranges or facets so that broad input coverage is not necessary.

It is used for finding the following errors:

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures or External database access
4. Performance errors
5. Initialization and termination errors

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 all levels of the software testing process: Unit Testing, Integration Testing, System Testing, and Acceptance Testing. The higher the level, and hence the bigger and more complex the box, the more black box testing method comes into use.

Black Box Testing Techniques

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

Equivalence partitioning

Equivalence Partitioning 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

Boundary Value Analysis is a software test design technique that involves 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

Cause Effect Graphing 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.

Comparison Testing
Different independent versions of same software are used to compare to each other for testing in this method.
Fuzz Testing
Fuzz testing is often employed as a black box software testing technique, which is used for finding implementation bugs using malformed / semi- malformed data injection in an automated or semi-automated fashion. Fuzzing is also used to test for security problems in software.
Model-based Testing
Model-based testing is automatic generation of efficient test procedures/vectors using models of system requirements and specified functionality. In this method, test cases are derived in whole or in part from a model that describes some aspects of the system under test. These test cases are known as the abstract test suite, and for their selection different techniques have been used.

Black Box Testing Advantages

- ☑ **Efficient Testing:** Well suited and efficient for large code segments or units.
- ☑ **Unbiased Testing**: clearly separates user's perspective from developer's perspective through separation of QA and Development responsibilities.
- ☑ **Non intrusive:** code access not required.
- ☑ **Easy to execute:** Test cases can be designed as soon as the specifications are complete. It can be scaled to large number of moderately skilled testers with no knowledge of implementation, programming language, operating systems or networks.
- ☑ **No Coding Skill Required:** Tester need not know programming languages or how the software has been implemented.

Black Box Testing Disadvantages

- ☒ **Localized Testing:** Limited code path coverage since only a limited number of test inputs are actually tested.
- ☒ **Inefficient Test Authoring**: without implementation information, exhaustive input coverage has unknown additional benefits to the actual code paths exercised and can require tremendous resources.
- ☒ **Blind Coverage**: cannot control targeting code segments or paths which may be more error prone than others
- ☒ **Redundant:** Tests can be redundant if the software designer/ developer has already run a test case.

2. White Box Testing

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. White Box Testing is contrasted with Black Box Testing.

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.

Levels Applicable To
It is applicable to the all levels of software testing: **1) Unit Testing:** For testing paths within a unit. **2) Integration Testing:** For testing paths between units. **3) System Testing:** For testing paths between subsystems. However, it is mainly applied to Unit Testing.

White Box Testing Techniques

Basis Path Testing
Basis path is a white box testing techniques which is first proposed by Tom McCabe and it allows the test case designer to produce a logical complexity measure of procedural design and use this measure as an approach for outlining a basic set of execution path (basic set is the set of all the execution of a procedure). These are test cases that exercise basic set will execute every statement at least once. Basic path testing makes sure that each independent path through the code is taken in a predetermined order.
Loop Testing
Loop testing is another type of white box testing which exclusively focuses on the validity of loop construct. Loops are simple to test unless dependencies exist between the
Loops or among the loop and the code it contain.
There are four classes of loops:

a) Simple Loop
b) Nested Loop
c) Concatenated Loop
d) Unstructured Loop

Control Structure Testing

The other synonym of control structure testing are conditional testing or decision testing or branch testing and comes under white box testing technique; it makes sure that each possible outcome from the condition is tested at least once. Branch testing however, has the objective to test every option (either true or false) on every control statement which also includes compound decision (when the second decision depends upon the previous decision). In branch testing, test cases are designed to exercise control flow branches or decision points in a unit. All branches within the branch are tested at least once.

White Box Testing Advantages

- ☑ **Increased Effectiveness:** Crosschecking design decisions and assumptions against source code may outline a robust design, but the implementation may not align with the design intent.
- ☑ **Full Code Pathway Capable:** all the possible code pathways can be tested including error handling, resource dependencies, and additional internal code logic/flow.
- ☑ **Early Defect Identification:** Analyzing source code and developing tests based on the implementation details enables testers to find programming errors quickly.
- ☑ **Reveal Hidden Code Flaws:** access to source code improves understanding and uncovering unintended hidden behavior of program modules.
- ☑ **No Waiting:** Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.

White Box Testing Disadvantages

- ☒ **Difficult to Scale:** requires intimate knowledge of target system, testing tools and coding languages, and modeling. It suffers for scalability of skilled and expert testers.
- ☒ **Difficult to Maintain:** It requires specialized tools such as source code analyzers, debuggers, and fault injectors.
- ☒ **Cultural Stress:** the demarcation between developer and testers starts to blur which may become a cultural stress.
- ☒ **Highly intrusive:** requires code modification has been done using interactive debuggers, or by actually changing the source code. This may be adequate for small programs; however, it does not scale well to larger applications. Not useful for networked or distributed systems.
- ☒ **Tools not readily available:** Since this method of testing it closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.
- ☒ **Mechanic Work:** White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

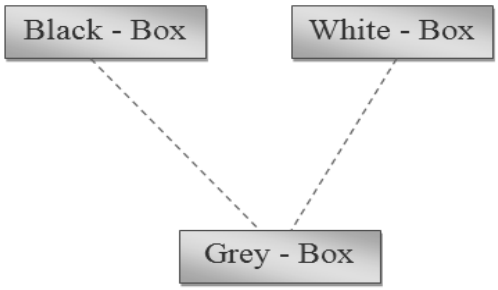


Fig 2: Gray Box Testing

3. Gray Box Testing

The aim of this testing is to search for the defects if any due to improper structure or improper usage of applications. Gray-box testing is also known as *translucent testing*.
Gray-box testing is well suited for **Web applications**.

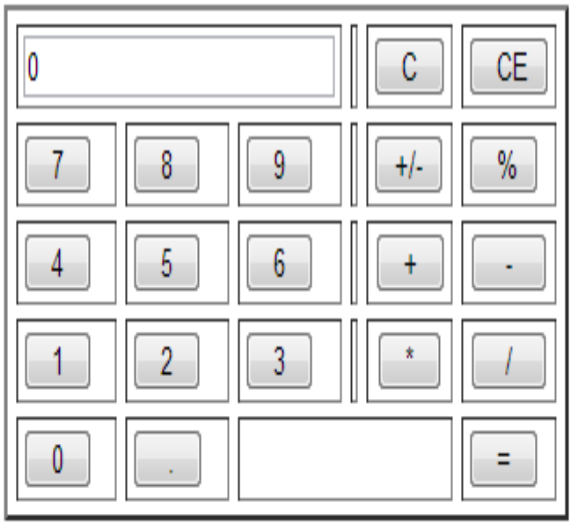
Web applications have distributed network or systems; due to absence of source code or binaries it is not possible to use white-box testing. Black-box testing is also not used due to just contract between customer and developer, so it is more efficient to use gray-box testing as significant information is available in Web Services Definition Language (WSDL).

Gray-box testing is suited for **functional or business domain testing**. Functional testing is done basically a test of user interactions with may be external systems .As gray-box testing can efficiently suits for functional testing due to its characteristics; it also helps to confirm that software meets the requirements defined for the software.

Example

Testing a calculator for the features like testing for all operations at the same time if any problem arises then we make a change in HTML code and check further so at the same we go for white box testing(changing codes in backend) and black box testing(testing features on front end).

E.g.: Html Calculator with code




```
<HTML>
<CENTER>
<FORM name="Keypad" action="">
<TABLE>
<B>
<TABLE border=2 width=50 height=60 cellpadding=1 cellspacing=5>
<TR>
<TD colspan=3 align=middle>
<input name="ReadOut" type="Text" size=24 value="0" width=100%>
</TD>
<TD
</TD>
<TD>
<input name="btnClear" type="Button" value="  C  " onclick="Clear()">
</TD>
<TD><input name="btnClearEntry" type="Button" value="  CE " onclick="ClearEntry()">
</TD>
</TR>
<TR>
<TD>
<input name="btnSeven" type="Button" value="  7  " onclick="NumPressed(7)">
</TD>
<TD>
<input name="btnEight" type="Button" value="  8  " onclick="NumPressed(8)">
</TD>
<TD>
<input name="btnNine" type="Button" value="  9  " onclick="NumPressed(9)">
```

```
</TD>
<TD>
</TD>
<TD>
<input name="btnNeg" type="Button" value=" +/- " onclick="Neg()">
</TD>
<TD>
<input name="btnPercent" type="Button" value="  % " onclick="Percent()">
</TD>
</TR>
<TR>
<TD>
<input name="btnFour" type="Button" value="  4  " onclick="NumPressed(4)">
</TD>
<TD>
<input name="btnFive" type="Button" value="  5  " onclick="NumPressed(5)">
</TD>
<TD>
<input name="btnSix" type="Button" value="  6  " onclick="NumPressed(6)">
</TD>
<TD>
</TD>
<TD align=middle><input name="btnPlus" type="Button" value="  +  " onclick="Operation('+')">
</TD>
<TD align=middle><input name="btnMinus" type="Button" value="   -   " onclick="Operation('-')">
</TD>
</TR>
<TR>
<TD>
<input name="btnOne" type="Button" value="  1  " onclick="NumPressed(1)">
</TD>
<TD>
<input name="btnTwo" type="Button" value="  2  " onclick="NumPressed(2)">
</TD>
<TD>
<input name="btnThree" type="Button" value="  3  " onclick="NumPressed(3)">
</TD>
<TD>
</TD>
<TD align=middle><input name="btnMultiply" type="Button" value="  *  " onclick="Operation('*')">
</TD>
<TD align=middle><input name="btnDivide" type="Button" value="   /   " onclick="Operation('/')">
</TD>
</TR>
<TR>
<TD>
<input name="btnZero" type="Button" value="  0  " onclick="NumPressed(0)">
</TD>
<TD>
<input name="btnDecimal" type="Button" value="   .  " onclick="Decimal()">
</TD>
<TD colspan=3>
</TD>
<TD>
<input name="btnEquals" type="Button" value="  =  " onclick="Operation('=')">
</TD>
</TR>
```

```
</TABLE>
</TABLE>
</B>
</FORM>
</CENTER>
<font face="Verdana, Arial, Helvetica" size=2>
<SCRIPT LANGUAGE="JavaScript">
<!-- Begin
var FKeyPad = document.Keypad;
var Accumulate = 0;
var FlagNewNum = false;
var PendingOp = "";
function NumPressed (Num) {
if (FlagNewNum) {
FKeyPad.ReadOut.value  = Num;
FlagNewNum = false;
   }
else {
if (FKeyPad.ReadOut.value == "0")
FKeyPad.ReadOut.value = Num;
else
FKeyPad.ReadOut.value += Num;
   }
}
function Operation (Op) {
var Readout = FKeyPad.ReadOut.value;
if (FlagNewNum && PendingOp != "=");
else
{
FlagNewNum = true;
if ( '+' == PendingOp )
Accumulate += parseFloat(Readout);
else if ( '-' == PendingOp )
Accumulate -= parseFloat(Readout);
else if ( '/' == PendingOp )
Accumulate /= parseFloat(Readout);
else if ( '*' == PendingOp )
Accumulate *= parseFloat(Readout);
else
Accumulate = parseFloat(Readout);
FKeyPad.ReadOut.value = Accumulate;
PendingOp = Op;
   }
}
function Decimal () {
var curReadOut = FKeyPad.ReadOut.value;
if (FlagNewNum) {
curReadOut = "0.";
FlagNewNum = false;
   }
else
{
if (curReadOut.indexOf(".") == -1)
curReadOut += ".";
   }
FKeyPad.ReadOut.value = curReadOut;
}
```

```
function ClearEntry () {
FKeyPad.ReadOut.value = "0";
FlagNewNum = true;
}
function Clear () {
Accumulate = 0;
PendingOp = "";
ClearEntry();
}
function Neg () {
FKeyPad.ReadOut.value = parseFloat(FKeyPad.ReadOut.value) * -1;
}
function Percent () {
FKeyPad.ReadOut.value = (parseFloat(FKeyPad.ReadOut.value) / 100) * parseFloat(Accumulate);
}
// End -->
</SCRIPT>
</HTML>
```

While you are testing the software, you need to give test cases in such a way that all the buttons in the GUI are tested. So, you need to generate the test cases. Below table show some test cases and expected result.

Test Case	Expected Output
1.2 * 3	3.6
5/ 2.0	2.5
7 + 8 - 9	6
600*2 %	12

Levels Applicable To

Though Gray Box Testing method may be used in other levels of testing, it is primarily useful in Integration Testing. It also includes in reverse engineering and testing database product.

Gray-Box Testing Techniques

1. Matrix Testing
2. Regression testing
3. Pattern Testing
4. Orthogonal Array Testing

Matrix Testing

It starts with developers defining all the variables that exist in their programs. Each variable will have an inherent technical risk, business risk and can be used with different frequency during it's' life cycle.

Regression testing

This testing is performed after making a functional improvement or repair to the program. Its purpose is to determine if the change has regressed other aspects of the program. This can be accomplished by executing the following testing strategies.

- → **Retest all:** Rerun all existing test cases
- → **Retest Risky Use Cases:** Choose baseline tests to rerun by risk
- → **Retest by Profile:** Choose baseline tests to rerun by allocating time in proportion to operational profile
- → **Retest Changed Segment:** Choose baseline tests to rerun by comparing code changes. (White box strategy)
- → **Retest within Firewall:** Choose baseline tests to rerun by analyzing dependencies. (White box strategy)

Pattern testing

It is best accomplished when historical data of previous system defects are analyzed. Your analysis will include specific reasons for the defect, which will require system analysis. Unlike black box testing where you are tracking the type of failures that are occurring, gray box testing digs within the code and determines why the failure happened. This information is extremely valuable,

ISSN 2277-1956/V2N1-175-185

as future design of test cases will be proactive in finding the other failures before they hit production. Remember, having coding structure in place influences gray box test case design. Therefore, take advantage of powerful technique and design test cases.

Analysis Template will include:

1. The problem addressed
2. Applicable situation
3. The problem that can be discovered
4. Related problems
5. Solution for developers that can be implemented
6. Generic test cases

New test cases will include:

1. Security related test cases
2. Business related test cases
3. GUI related test cases
4. Language related test cases
5. Architecture related test cases
6. Database related test cases
7. Browser related test cases
8. Operating System related test cases

Orthogonal Array Testing

Orthogonal Array Testing is a statistical testing technique implemented by Taguchi. This method is extremely valuable for testing complex applications and e-comm products. The e-comm world presents interesting challenges for test case design and testing coverage. The black box testing technique will not adequately provide sufficient testing coverage.

The underlining infrastructure connections between servers and legacy systems will not be understood by the black box testing team. A gray box testing team will have the necessary knowledge and combined with the power of statistical testing, an elaborate testing net can be set-up and implemented.

The Orthogonal Array Testing (OAT) can be used to reduce the number of combinations and provide maximum coverage with a minimum number of test cases. OAT is an array of values in which each column represents a variable - factor that can take a certain set of values called levels. Each row represents a test case. In OAT, the factors are combined pair-wise rather than representing all possible combinations of factors and levels.

Gray-Box testing Advantages

- ☑ **Offers Combined Benefits:** Leverage the strengths of both Black Box and White Box testing, wherever possible.
- ☑ **Non Intrusive:** Gray Box does not rely on the access to source code or binaries. Instead, is based on interface definitions, functional Specifications and application architecture.
- ☑ **Intelligent Test Authoring:** Based on the limited information available, a Gray Box tester can author intelligent test scenarios, especially around data type handling, communication protocols, and exception handling.

- ☑ **Unbiased Testing:** The demarcation between testers and developers is still maintained. The handoff is only around interface definitions and documentation, without access to source code or binaries.

Gray-Box testing Disadvantages

- ☒ **Partial Code Coverage:** since the source code or binaries are not available, the ability to traverse code paths is still limited by the tests deduced through available information. The coverage depends on the tester authoring skills.
- ☒ **Defect Identification:** inherent to distributed applications is the difficulty associated in defect identification. Gray box testing is still at the mercy of how well systems throw exceptions, and how well are these exceptions propagated with a distributed Web Services environment.

Bridge between Black Box and White Box – Gray Box Testing Technique

Comparison between the Three Testing Types

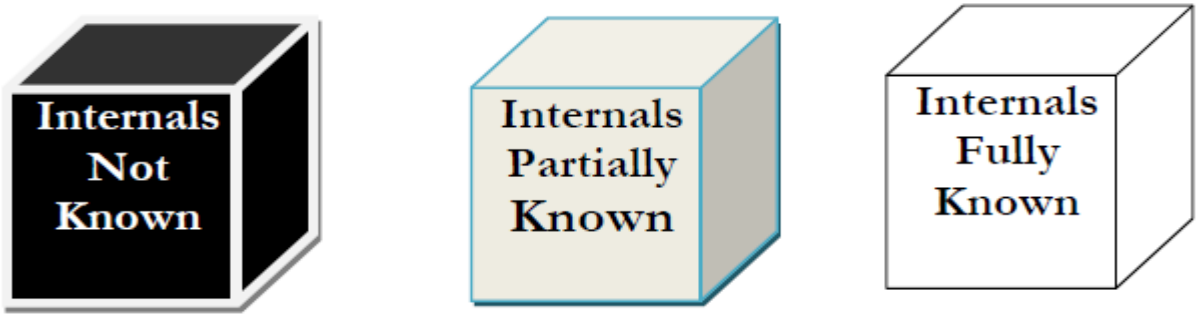


Fig-3: Comparison of Box Testing

S.N.	Black Box Testing	Gray Box Testing	White Box Testing
1	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2	Also known as closed box testing, data driven testing and functional testing	Another term for Gray box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4	Testing is based on external expectations - Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
7	This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

Future scope

For most applications that now are under development, white and black box testing was defined long ago and does not cover all the possible scenarios that must be tested. By implementing gray box testing, we simply reduce the overall cost of system defects - and prevent more from passing the testing stage. Precisely, if the process is based on the effectiveness of the black-box testing, as a "non-biased" and "agnostic" method, then gray-box testing will work against that. But if the development process tries to accumulate the benefits of both black-box and gray-box (for example, some of the members of the testing team can use exclusively black-box testing, while others are also allowed to rely on white-box), then gray-box testing is also a good option.Gray-box testing is well suited for Web applications, Web services, functional or business domain testing, security assessment, GUI, distributed environments, etc.

References

[1] www.scribd.com/doc/7142992/Gray-Box-Testing

[2] www.software**testing**class.com/gray-**box-testing**

[3] www.site.uottawa.ca/~ssome/Cours/SEG3203/g**boxtesting**.pdf

[4] https://www.evernote.com/shard/s2/note/c8dad139-d02d.../shared

[5] www.careerride.com/Testing-white-box-black-box-gray-box.aspx

[6] Software Engineering by Ivan Marsic - Rutgers University, 2008

[7] The New Software Engineering by Sue Conger - Global Text Project, 2008

[8] R. Pressman: Software Engineering, McGraw-Hill.

[9]K.K. Agrawal and Y. Sing: Software Engineering, New Age International.

[10]P. Jalote: Software Project Management in Practice, Pearson.