MANTECH Publications

Journal of Software Engineering & Software Testing
Volume 7 Issue 2
ISSN: 2457-0516 (Online)

# *Various Strategies for Employing the Black Box Method for Error Detection*

***Dr. P. N. Murthy[1], S. Laxmi[2]***
*Professor[1], Student[2]*
*Department of Information Technology*
*Krupajal Engineering College*
***Corresponding Author's Email: -*** *laxmi.s014@gmail.com[2]*

***Abstract***

*The technique of assessing software to determine the difference between needed and current conditions is known as software testing. Software testing is conducted throughout the software development cycle and is also undertaken to generate quality software; two main testing methodologies are utilised for this purpose: white box testing and black box testing. One of the software testing techniques that I have explained in my paper is Black Box Testing, which is a method of generating test cases that are independent of the internal structure of the software. I have also briefly explored various approaches to the black box testing technique for finding errors. Because black box testing is always dependent on the software specification, either directly or indirectly, it is also known as specification based testing.*

***Keywords:*** *Partitioning by Equivalence, Boundary Value Analysis, Fuzz Testing, Orthogonal Array Testing, and All Pair Testing*

## INTRODUCTION

Black box testing and white box testing are the two primary techniques of software testing. White box testing is based on an examination of a piece of software's internal workings and structure. It solely examines how the system processes the input to produce the desired outcome. Black box testing, on the other hand, focuses on the software's functional requirements. Black box testing is an essential component of correctness testing, although its concepts are not restricted to correctness testing. Black box testing is a


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

supplement to white box testing and is more likely to find a different type of fault than white box testing. In black box testing, the tester just knows about the input (processed by a system) and needed output; in other words, the tester does not need to know about the internal workings of the system. Black box testing occurs at several stages of the software development and testing life cycles, including regression testing, acceptance testing, unit testing, integration testing, and system testing. This technique's sorts of testing are entirely focused on testing the functioning of software applications.

Other terms for BBT include "opaque testing," "functional testing," "behavioural testing," and "closed-box testing." A search engine is a great example of a BBT system; we enter text that we want to search for and get the results, but we don't know or see the particular process that is being used to get our results. (E.g. Input ☐ search ☐ Output) (No. internal process)

Black box testing tools are mainly record and playback tools which record test cases in the form of some scripts like Perl, TSL, VB script, JAVA script. Selecting a black box test tool can be a challenging task due to the wide array of available commercial vendors and open source initiatives in this field These tools are used for regression testing and determining whether a fresh build has introduced any bugs into previously functioning application functionality. [1]

Before deciding on a black box tool for our application, we must examine a number of high-level factors, including:

- Ease of use
- Cost
- Accuracy
- Test coverage
- Test completeness
- Reporting capability
- Capacity of vulnerability database

There are certain pros and cons of Black Box Testing [2][3][4]

**Pros**

For starters, because the designer and tester are independent of one another, the test is neutral. Second, the test is conducted from the standpoint of the user rather than the standpoint of the designer. Third, it is simpler for testers to construct test cases by just navigating the programme like an end user would. Fourth, because the tester is not concerned with the inner workings of an application, they do not require knowledge of any specific programming language. Fifth, white box testing is more effective on larger pieces of code. Finally, faster test


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

case development because the tester is just concerned with the graphical user interface.

**Cons**

To begin with, script maintenance is quite tough since black box tools are dependent on the manner of input being understood. Second, without a clear and precise specification, it is impossible to develop test cases. Third, if the tester is not told of test scenarios that the programmer has previously tested, the test may be redundant. Finally, numerous programme paths may go unexplored.

**WORKING PROCESS OF BLACK BOX TESTING TECHNIQUE**

**The steps below describe the operation of Black Box Testing.**

**Step 1 Input:** The system's requirements and functional specifications are analyzed. High-level design documentation and source code for application blocks are also inspected.

The tester selects valid input and rejects incorrect input.

**Step 2 Processing Unit:** Do not be concerned with the system's internal workings. In the processing unit, the tester creates and executes test cases based on the given input.

Load testing, stress testing, security auditing, and globalization testing are also performed by the tester. If a flaw is discovered, it will be repaired and retested.

**Step 3 Output:** After all of this testing, the tester obtains the expected results and provides the final report.

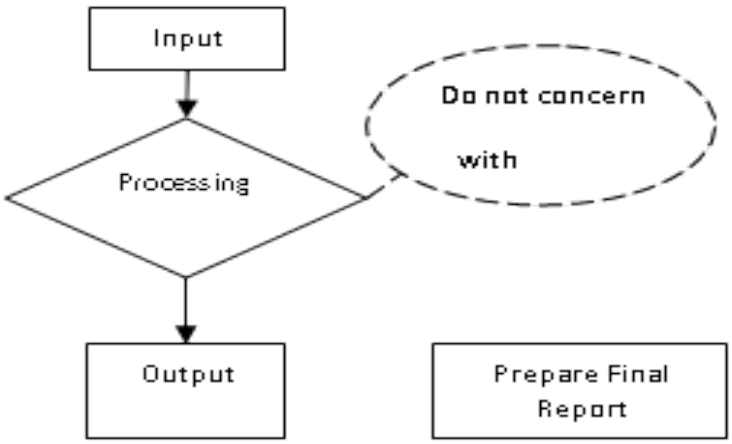



***Figure 1. Working process of black box testing technique***


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

**Different Forms of Black Box Testing Technique Are**

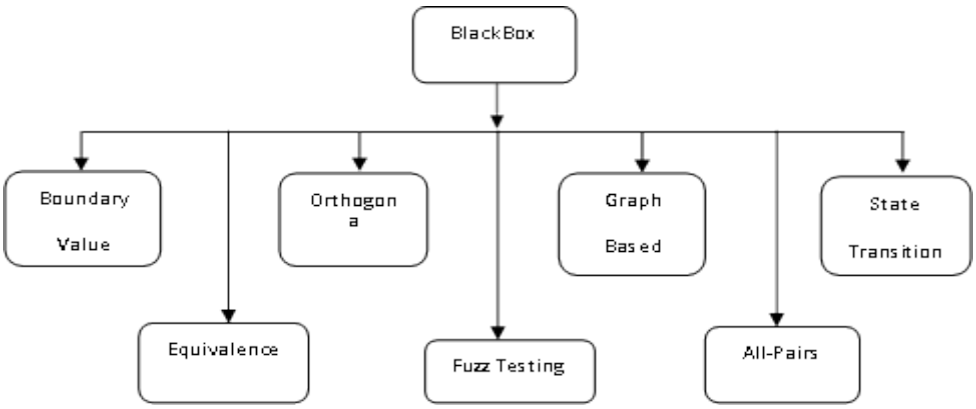



***Figure 2. Various forms of black box testing techniques***

**Equivalence Partitioning**

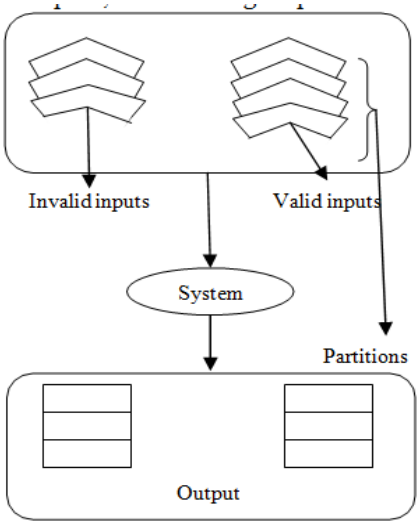



***Figure 3 Equivalence class partitioning***

Equivalence partitioning is a black box testing approach that splits a software unit's input data into data partitions from which test cases may be constructed. It decreases the number of test cases.

An equivalence class is produced in equivalence class partitioning by the inputs for which the system's behaviour is stated or anticipated to be comparable. A collection of valid or incorrect states for input conditions is represented by an equivalence class.

An input condition is typically a specific numeric value, an array of values, a group of related values, or a Boolean condition. After we've chosen equivalence classes for each of the inputs, the following step is to choose appropriate test cases. [5]


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

Some of the guidelines for equivalence partitioning are given below: [6]

1. One valid and two invalid equivalence classes are defined if an input condition specifies a range.

2. One valid and two invalid equivalence classes are defined if an input condition requires a specific value.

3. One valid and one invalid equivalence class are defined if an input condition specifies a no. of a set.

4. One valid and one invalid equivalence class are defined if an input condition is Boolean. Now, let us consider an example of equivalence class partitioning:

Table 1 shows an example set of legal and invalid equivalence classes for a programme that takes two inputs, a string 'C' of length up to M and an integer 'P'. The application will discover the top P most often appearing character in C. [5]

***Table 1. Valid & Invalid Equivalence Classes for the Above Example***

| Input | Valid E C | Invalid E C |
|---|---|---|
| C | 1:Contains numbers<br>2:Contains lower case letter<br>3:Contains upper case letter4:Contains special character<br>5:Sting length between O - M | 1:Non-ASCII character<br>2:String length □M |
| P | 6:Integer in valid range | 3:Integer out of range |

**Boundary Value Analysis**

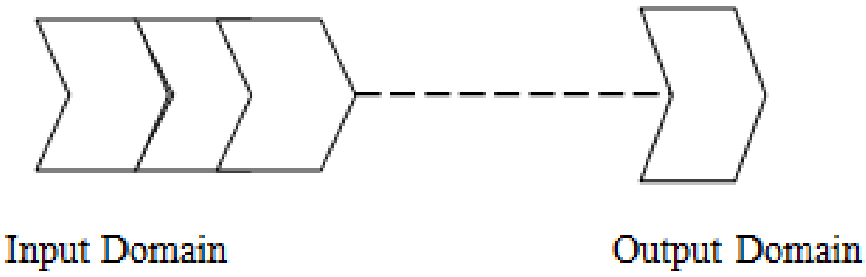



***Figure 4 Boundary value analysis***


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

"Bugs hide in corners and cluster at borders," which means that more systems fail on the boundary because programmers frequently make mistakes on the boundary of the equivalence classes/input domain. Boundary value analysis is a testing approach that focuses on testing at borders or at extreme boundary values. Maximum, minimum, slightly inside/outside borders, usual values, and error values are all examples of boundary values.

Assume that in boundary value analysis, each input value has a predetermined range, then the following six boundary values are given: [5]

1. The extreme ends of the range.
2. Just beyond the ends.
3. Just before the ends.

i.e. if an integer range is min to max, then six values are

a. Min-1
b. Min
c. Min+1
d. Max-1
e. Max
f. Max+1

Now, I will explain boundary value analysis with the help of an example:

E.g. 1: if input condition have a range from a to v (a=10 to b=60), create test cases

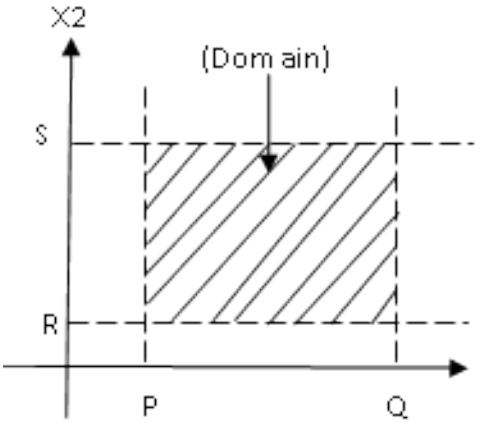
1. Immediately below a (min-1)=9
2. At a (min)=10
3. Immediately above a (min+1)=11
4. Immediately below b (max-1)=59
5. At b (max)=60
6. Immediately above b (max+1)=61

**E.g. 2:** Suppose X1 and X2 are two variables where X1 lies between P & Q and X2 lies between R & S

$P \leq X1 \leq Q$

$R \leq X2 \leq S$

Values P, Q, R, and S are extremities of the input domain which are demonstrated below: [7]




There are two ways to generate boundary value analysis technique

1. By the no. of variables (for n variables)
2. By the kinds of ranges

One of the important drawback of BVA is that it is only efficient for variable of fixed values i.e. boundary


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

**Fuzzing**

Fuzz testing is frequently used as a black box software testing approach for detecting implementation issues through automated or semi-automated malformed / semi-malformed data injection. Fuzzing is also used to evaluate software for security flaws.

**Two forms of fuzzing programs are**

1. Mutation based-mutation based fuzzers mutate existing data sample to create test data.
2. Generation based- generation based fuzzers define new test data based on models of input.

Fuzzing can also advise which parts of the programme should be given more attention, such as through a code audit, static analysis, or partial rewrites. Fuzz testing is used to detect problems like assertion failures and memory leaks. Protocol fuzzing involves sending forged packets to the tested application, altering requests on the fly, and replaying them—two constraints.

1. Testing cannot proceed until the specification is relatively mature.
2. Many useful protocols are proprietary.

One of the most significant benefits of fuzz testing is that the test designs are incredibly simple to understand and devoid of preconceived notions about system behaviour. The biggest problem of using fuzzing to identify programme errors is that it often only finds relatively simple errors. Another drawback of fuzzing is that anytime we do black box testing, we generally have a closed system to attack, which makes evaluating the impact of the discovered vulnerability difficult, i.e. debugging is not feasible. [8]

**Cause Effect Graph**

It is a black box testing approach in which the testing process begins with the creation of a graph and the establishment of the relationship between the effect and its causes. Where "cause" is an input condition and "effect" is a calculation sequence to be done. Each condition is represented by a node in the cause effect graph. Cause effect graphs are essentially directed graphs with four core symbols expressing the interdependence of cause and effect. [5] [9]

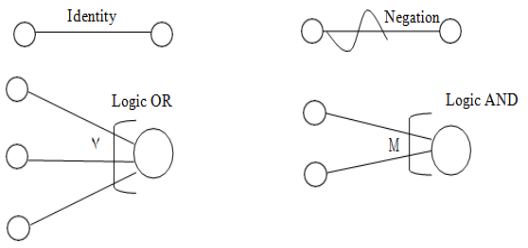



***Figure 5. Basic symbols of cause effect graph***


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

Let us consider an example showing cause effect graph and the decision table.

For example, the first column's character should be "P" or "Q." The second column must include a number. If the initial character is incorrect, the message A12 should be printed. If there was no number in the second column, reprint the message A13.

N1: update the file

N2: message A12

N2: message A13

P1: first character is “P”

P2: the first character is “Q”

P3: the second is a number

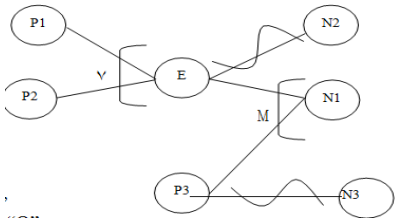



***Figure 6. Cause effect graph for the above example***

**Table 2. Cause effect graph decision 1=true 0=false**

| Test Data | Causes | | | Effect | | |
|---|---|---|---|---|---|---|
| | P1 | P2 | P3 | N1 | N2 | N3 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 1 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 1 | 0 | 0 |

3 input, 8 possibilities But P1=true and P2=true is an impossible condition

Cause effect graphing, before generating high, just test cases, also aids the understanding of the functionality of the system, because the tester must identify the distinct causes and effect

**Orthogonal Array Testing**

Before we can discuss orthogonal array testing, we must first define orthogonal array; it is a 2-dimensional property that gives any pair combination of each number in the array by selecting any two columns in the array. [10] OAT is a black box testing approach that may be used to solve situations where the input domain is relatively small but too vast for exhaustive testing. [11] System testing, user interface testing, regression testing, configuration testing, and performance testing all employ OAT.

Each column in OAT represents a variable, whereas each row represents a test case. Rather of expressing every potential combination of elements and levels, OAT combines them pairwise. OAT allows us to create test cases that give maximum test coverage while using a manageable amount of test cases. [5]


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

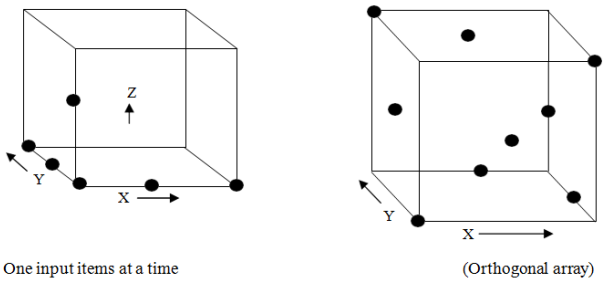



***Figure 7. Geometric view of test cases***

OAT has several advantages, including reduced testing cycle time, evenly distributed coverage of the test domain, and fewer test cases created due to a compact test collection. Despite all of its advantages, OAT has one drawback: it does not ensure comprehensive coverage of the test domain. [5]

**All-Pair Testing**

It is Black box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters. In pair-wise testing we have to exercise all pairs of values during testing. [10]

As if there are 'a' parameters, each with 'b' values then between each two parameter we have a * b pair. The main objective of pair wise testing is to have a set of test cases that covers all the pairs. As is there are 'p' parameters, then the set of test cases will cover (p-1) + (p-2) +

= p (p-1)/2 pairs.

Now let us consider an example of all pair testing E.g. suppose there are three parameters A (secondary memory), B (primary memory) and C (operating system). Each of them can have three values (a1, a2, a3), (b1, b2, b3), (c1, c2, c3). Then the total number of pair wise combination is 9*3=27, these test cases are shown in table below: [5]

As no testing technique can find all bugs, pair wise testing is typically used together with other quality assurance techniques such as fuzz testing, unit testing and code review.

**Table 3. Test cases of all pair testing**

| A | B | C | Pairs |
|---|---|---|---|
| a1 | b1 | c1 | (a1, b1) (a1, c1) (b1, c1) |
| a1 | b2 | c2 | (a1, b2) (a1, c2) (b2, c2) |
| a1 | b3 | c3 | (a1, b3) (a1, c3) (b3, c3) |
| a2 | b1 | c2 | (a2, b1) (a2, c2) (b1, c2) |
| a2 | b2 | c3 | (a2, b2) (a2, c3) (b2, c3) |
| a2 | b3 | c1 | (a2, b3) (a3, c1) (b3, c1) |
| a3 | b1 | c3 | (a3, b1) (a3, c3) (b1, c3) |
| a3 | b3 | c1 | (a3, b3) (a3, c1) (b3, c1) |
| a3 | b3 | c3 | (a3, b3) (a3, c3) (b3, c3) |


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

**State Transition Testing**

This is another type of black box testing that is effective for evaluating state machines as well as graphical user interface navigation. Delegate will create a state transition diagram and test different degrees of coverage. A different component known as a state table can also be used to produce negative tests ( a grid showing the result ling transitions for each state combine with each possible event, showing both valid and invalid transition). [10]

If we have a reasonable number of states for a system, we may design a state model, which has four key consequences: [5]

1. States → how the state of the system changes from one state to another.
2. Transition→ how the state of the system changes from one state to another.
3. Events→input to the system
4. Actions→output for the events

E.g. An example of state transition testing for a machine M with corresponding state diagram and state transition table.

Table 4. State transition

| State input | 1 | 0 |
|---|---|---|
| S1 | S1 | S2 |
| S2 | S2 | S1 |

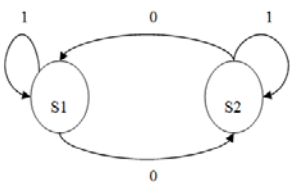



***Figure 8. State diagram***

State transition testing may thus be defined as a black box test design in which test cases are created to perform valid and invalid state transitions.

**The Current Scenario in Black Box Testing**

According to recent reports, the executive director of pure water house coopers and other executive directors of other enterprises recommends that the phenomenon of black box testing be adopted early in the product life cycle. Today, every industry employs black box testing. It is a component of the Capability Maturity Model Integration (CMMI) and is utilised in a variety of industries including FMCG, retail, and aviation. The black box is here to stay, but developers should put in more effort to understand programming in order to do their jobs more efficiently.

**CONCLUSION**

I conclude that black box testing is a testing approach that ignores a system's fundamental mechanism or structure and concentrates on the outputs created in response to certain inputs and execution circumstances. Black box testing is used to assess a system's compliance with stated functional criteria and the related projected outputs.


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

**The various approaches to black box testing which I have described in my paper are:**

1. **Equivalence portioning:** This method separates the input data into partitions from which test cases may be constructed.

2. **Boundary value analysis:** This type of analysis focuses on testing at boundaries or in areas where the extreme boundary values are chosen.

3. **Fuzzing:** It is used to uncover implementation bugs as well as to test for security issues in software.

4. **Create a cause:** -effect graph and draw a relationship between the effect and the causes.

5. **Orthogonal array testing:** It can be used to solve problems where the input domain is tiny but too large for exhaustive testing.

6. **All pair testing:** In this test scenario, all conceivable discrete combinations of each pair of input parameters are executed.

7. **State transition testing:** In this test scenario, correct and invalid state transitions are executed.

**REFERENCES**

1. Orthogonal Testing available at http://mytestingexp.wordpress.com /2010/08/05/orthogonal- testing-and-pairwise-testing/

2. The Pros and Cons of Black Box Testing Technique available at http://www.testplant.co download_files/BB_vs_WB_testing.pdf

3. Black Box Test Tool available at https://buildsecurityin.us-cert.gov/bsi/articles/tools/black-box/261-BSI.html

4. Advantages and Disadvantages of Black Box Testing available at http://www.geekinterview.com/blogs/243-advantages-and-disadvantages-black-box-testing.html

5. Boundary Value Analysis by Blake Neate (327966) availableathttp://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NeateB.pdf


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved

6. Black Box Testing available at http://www.webopedia.com/TERM/B/Black_Box_Testing.html

7. An Integrated Approach to Software Engineering (Third Edition) by Pankaj Jalote, published in Narosa Publishing House Pvt. Ltd.

8. Black Box Testing available at http://www.softwaretestinghelp.com/black-box-testing/

9. Fuzzing available at https://www.owasp.org/index.php/Fuzzing

10. Example of Cause Effect Graph proposed by G. J. Mayers on 13/9/2007

11. Standard glossary of terms used in Software Testing (ISTQB) version 2.1 (dd. April 1st, 2010) by Erik van Veenendaal (The Netherlands)

12. The new age of Black Box Testing available at http://www.globalservicesmedia.com/IT- Outsourcing/Product-Development/The-New-Age-of-Black-Box-Testing/22/4/0/GS100208218035


Page 85-96 © MANTECH PUBLICATIONS 2022. All Rights Reserved