International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.4, October 2011

DIFFERENT APPROACHES TO BLACK BOX TESTING TECHNIQUE FOR FINDING ERRORS

Mohd. Ehmer Khan

Department of Information Technology
Al Musanna College of Technology, Sultanate of Oman
ehmerkhan@gmail.com

ABSTRACT

Software testing is the process of analyzing software to find the difference between required and existing condition. Software testing is performed throughout the development cycle of software and it is also performed to build quality software, for this purpose two basic testing approaches are used, they are white box testing and black box testing. One of the software testing technique which I have explain in my paper is Black Box Testing, it is a method of generating test cases that are independent of software internal structure, I have also briefly explore various different approaches to black box testing technique for finding errors. Since black box testing is always based either directly or indirectly on the software specification so it is also called specification based testing.

KEYWORDS

Equivalence Partitioning, Boundary Value Analysis, Fuzz Testing, Orthogonal Array Testing, All Pair Testing

1. INTRODUCTION

Two basic approaches to software testing are black box testing and white box testing. White box testing based on an analysis of internal working and structure of a piece of software. It only checks how the system processes the input to generate required output. On the other hand black box testing focuses on the functional requirement of the software. Black box testing is an integral part of correctness testing but its ideas are not limited to correctness testing only. Black box testing is complementary to white box testing technique and likely to uncover a different class of errors then white box method.

The tester, in black box testing only knows about the input (process by a system) and required output, or in the other word tester need not know the internal working of the system.

Black box testing occur throughout the software development life cycle and software testing life cycle i.e. in regression testing, acceptance testing, unit testing, integration testing and system testing stage. The types of testing under this technique are totally focused on the testing for functionality of the software applications.

The other synonyms of BBT are “opaque testing”, “functional testing”, “behavioral testing” and closed-box testing”. An ideal example of BBT system would be a search engine, in which we enter text that we want to search for and got the result, we do not know or see the specific process that is being employed to obtain our result.

(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

DOI : 10.5121/ijsea.2011.2404

projects in this area. These tools are used for regression testing and to check whether new build has created any bug in previous working application functionality. [1]

Before selecting the black box tool for our application we have to contemplate a no. of high level consideration such as: [1]

- 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

Firstly, as the designer and tester are independent of each other the test is unbiased. Secondly, the test is not done from the point of view of designer it is rather done from the point of view of user. Thirdly, it is easier for tester to create test cases by simply working through the application, as would an end user. Fourthly, the tester does not need knowledge of any specific programming language as they do not have to concern themselves with the inner working of an application. Fifthly, more effective on larger units of code than white box testing. Lastly, quicker test case development as the tester only concerns them with the graphical user interface.

Cons

Firstly, script maintenance is very difficult as black box tools are relevant on the method of input being known. Secondly, test cases are difficult to design without clear and concise specification. Thirdly, the test can be redundant if the tester is not informed of test cases that programmer has already tried. Lastly, may leave many program path untested.

2. WORKING PROCESS OF BLACK BOX TESTING TECHNIQUE

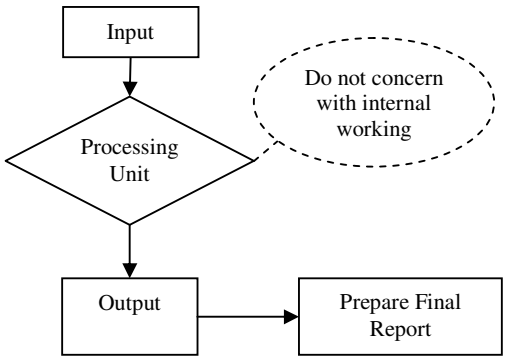


Figure 1. Working process of black box testing technique

Below are the steps which explain the working process of Black Box Testing

Step 1 Input: Requirement and functional specification of the system are examined. High level design documents and application block source code are also examined. Tester chooses valid input and rejects the invalid inputs.

Step 2 Processing Unit: Do not concern with the internal working of the system. In processing unit tester constructs test cases with the selected input and execute them. Tester also performs

load testing, stress testing, security review and globalization testing. If any defect is detected it will be fixed and re-tested.

Step 3 Output: After all these testing, tester gets desired output and prepares final report.

3. DIFFERENT FORMS OF BLACK BOX TESTING TECHNIQUE ARE

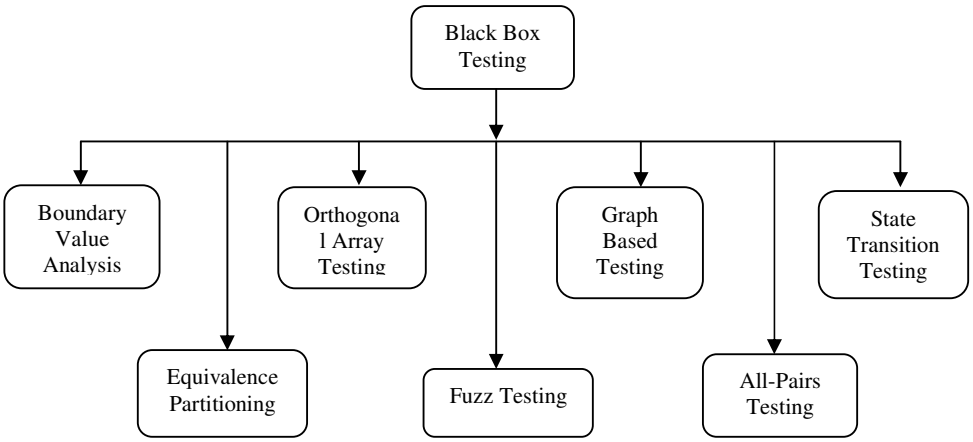


Figure 2. Various forms of black box testing techniques

3.1 Equivalence Partitioning

Equivalence partitioning is a black box testing method that divides the input data of a software unit into partitions of data from which test cases can be derived. It reduces no. of test cases. In equivalence class partitioning an equivalence class is formed of the inputs for which the behavior of the system is specified or expected to be similar. An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric values, array of values, a set of related values or Boolean condition. Once we select equivalence classes for each of the input, the next issue is to select the test cases suitably. [5]

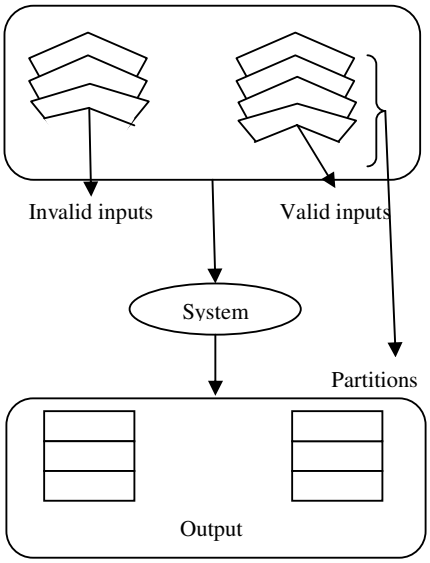


Figure 3. Equivalence class partitioning

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:

Example set of valid and invalid equivalence classes are shown below in table 1for a program that takes two inputs, a string 'C' of length up to M and integer 'P'. The program is determine the top P highest occurring 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 2:Contains lower case letter 3:Contains upper case letter 4:Contains special character 5:Sting length between O - M	1:Non-ASCII character 2:String length>M
P	6:Integer in valid range	3:Integer out of range

3.2 Boundary Value Analysis

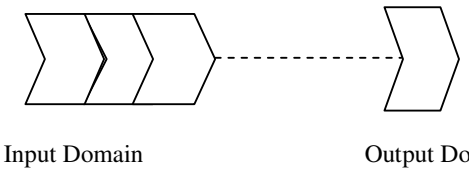


Figure 4. Boundary value analysis

"Bug lurk in corners and congregate at boundaries" by this we mean more systems have tendency to fail on boundary as programmers often make mistakes on the boundary of the equivalence classes/input domain. Boundary value analysis is a testing technique that focuses more on testing at boundaries or where the extreme boundary values are chosen. Boundary value include maximum, minimum, just inside/outside boundaries, typical values and error values.

Suppose in boundary value analysis each input values has defined range, than there are six boundary values which are given below: [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

1) Min-1
2) Min

3) Min+1
4) Max-1
5) Max
6) Max+1

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

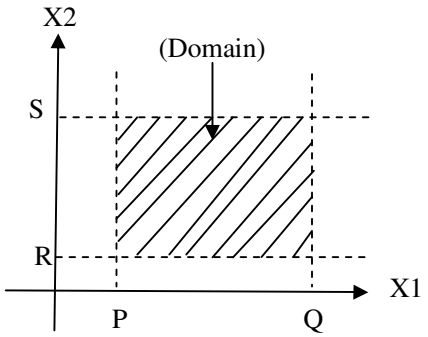
E.g. 1: if input condition have a range from a to v (a=10 to b=60), create test cases

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

3.3 Fuzzing

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.

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 suggest which part of program should get special attention, in the form of a code audit, application of static analysis, or partial rewrites. Fuzz testing is used to find bugs such as assertion failure and memory leaks. In protocol fuzzing a protocol fuzzer sends forged packets to the tested application, modifying request on the fly and replaying them-two limitations.

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

One of the most important advantages of fuzz testing is that the test designs are extremely simple to understand and are free of pre conception about system behavior. The main limitation with fuzzing to find program fault is that it generally only finds very simple faults. Another limitations with fuzzing is that whenever we perform any black box testing we usually have a closed system to attack, which increases difficulty to evaluate the impact of the found vulnerability i.e. debugging is not possible. [8]

3.4 Cause Effect Graph

It is black box testing technique in which testing begins by creating a graph and establishing the relation between the effect and its causes. Where "cause" is an input condition and "effect" is a sequence of computations to be performed. Each condition forms a node in the cause effect graph. Cause effect graph is basically a directed graph and there are four basic symbols express the interdependency between cause and effect. [5][9]

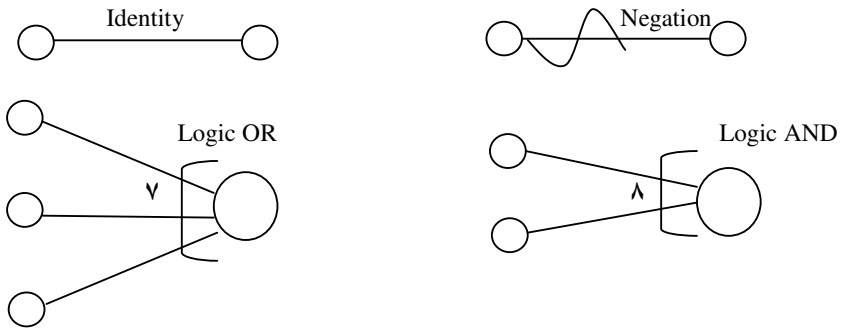
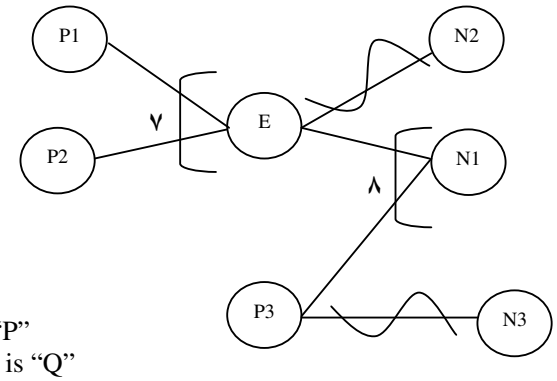


Figure 5. Basic symbols of cause effect graph

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

E.g. the character of a first column should be "P" or "Q". The second column should be a number. If the first character is erroneous, we should print the message A_{12}. If the second column we didn't have a number, reprint the message A_{13}.



N1: update the file
N2: message A_{12}
N2: message A_{13}

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

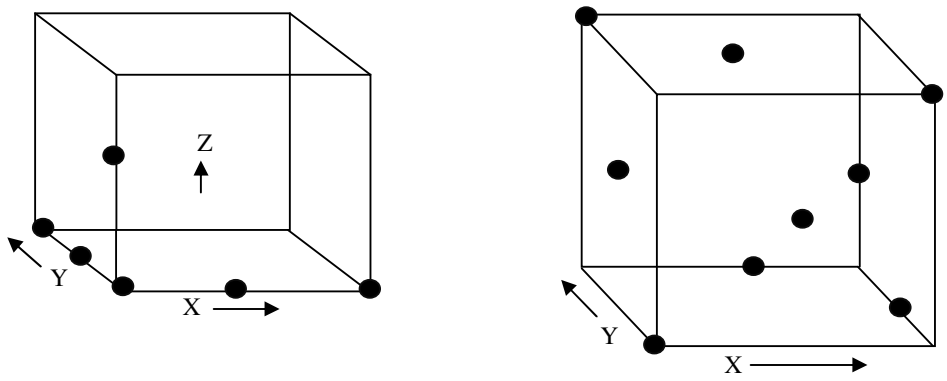
Cause effect graphing, before generating high, just test cases, also aids the understanding of the functionality of the system, because the tester must identify the distinct causes and effect

3.5 Orthogonal Array Testing

Before explaining orthogonal array testing, we must understand the meaning of orthogonal array; it is a 2-dimensional property, such that choosing any two columns in the array provides every pair combination of each number in the array. [10] OAT is a black box testing techniques which can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing. [11] OAT applied in system testing user interface testing, regression testing, configuration testing and performance testing.

In OAT each column represents a variable and each row represents a test case. Rather than representing all possible combination of factors and levels, in OAT the factors are combined pair wise. OAT enables us to design test cases that provide maximum test coverage with reasonable number of test cases. [5]

There are certain benefits of OAT such as it reduces testing cycle time, provides uniformly distributed coverage of the test domain, fewer test cases are generated because concise test set. With all the benefits there is one limitation with OAT that is; it does not guarantee the extensive coverage of test domain. [5]



One input items at a time (Orthogonal array)

Figure 7. Geometric view of test cases

3.6 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]

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)

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.

3.7 State Transition Testing

This is another black box testing which is useful when testing state machines and also navigation of graphical user interface. Delegate will generate state transition diagram and will test various levels of coverage. Negative tests can also be generated using a different aspect called a state table (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 manageable set of states for a system then we can able to build state model which has four important impacts: [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

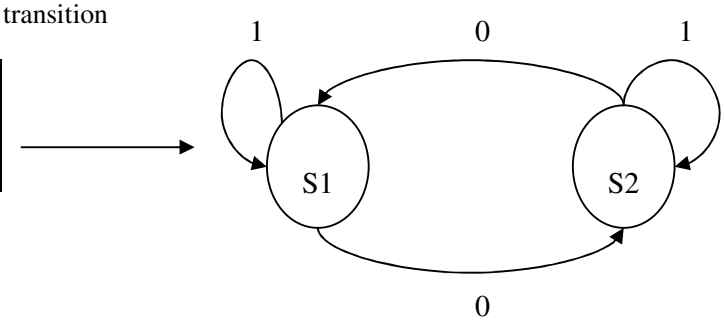


Figure 8. State diagram

So we can define state transition testing as a black box test design in which test cases are designed to execute valid and invalid state transition.

4. THE CURRENT SCENARIO IN BLACK BOX TESTING

According to the recent reports, the executive director with pure water house coopers in various other executive directors of different companies suggests that the phenomena of black box testing should be introduced early in the product life cycle. Today very sector uses black box testing. It is an integral part of the Capability Maturity Model Integration (CMMI) and is used across all sectors such as FMCG, Retail, and Aviation etc. Black box is here to stay but developer should make more effort to learning programming for efficient work. [12]

5. CONCLUSION

I conclude black box testing is a testing technique that ignores the internal mechanism or structure of a system and focuses on the outputs generated in response to selected inputs and execution conditions. Black box testing is conducted to evaluate the compliance of a system with specified functional requirements and corresponding predicted results.

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

1. Equivalence portioning: It divides the input data into partitions of data from which test cases can be derived.
2. Boundary value analysis: It focus more on testing at boundaries or where the extreme boundary values are chosen.
3. Fuzzing: It is used for finding implementation bug and also used to test for security problem in software.
4. Cause effect graph: Graph is created and a relation between the effect and causes are established.
5. Orthogonal array testing: It can be applied to problem in which the input domain is relatively small but too large to accommodate exhaustive testing.
6. All pair testing: In this test cases are designed to execute all possible discrete combinations of each pair of input parameters.
7. State transition testing: in this test cases are designed to execute valid and invalid state transitions.

REFERENCES

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

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

[3] The Pros and Cons of Black Box Testing Technique available at http://www.testplant.com/download_files/BB_vs_WB_testing.pdf

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

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

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

[7] Boundary Value Analysis by Blake Neate (327966) available at http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NeateB.pdf

[8] Fuzzing available at https://www.owasp.org/index.php/Fuzzing

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

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

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

[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

Mohd. Ehmer Khan

I completed my B.Sc in 1997 and M.C.A. in 2001 from Aligarh Muslim University, Aligarh, India, and pursuing Ph.D (Computer Science) from Singhania University, Jhunjhunu, India. I have worked as a lecturer at Aligarh College Engineering & Management, Aligarh, India from 1999 to 2003. From 2003 to 2005 worked as a lecturer at Institute of Foreign Trade & Management, Moradabad, India. From 2006 to present working as a lecturer in the Department of Information Technology, Al Musanna College of Technology, Ministry of Manpower, Sultanate of Oman. I am recipient of PG Merit Scholarship in MCA. My research area is software engineering with special interest in driving and monitoring program executions to find bugs, using various software testing techniques.