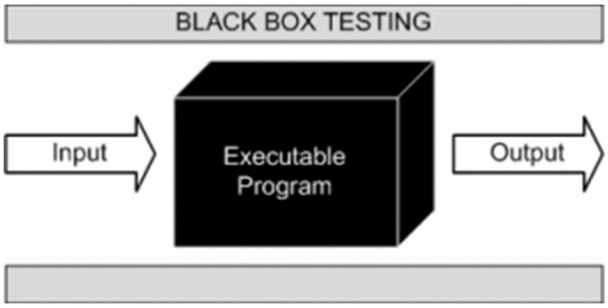
BLACK BOX TESTING

BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.

This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

Techniques

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

- *Equivalence Partitioning:* It 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:* It is a software test design technique that involves the 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:* It 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.

Advantages

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

TYPES OF BLACK BOX TESTING

Practically, due to time and budget considerations, it is not possible to perform exhausting testing for each set of test data, especially when there is a large pool of input combinations.

- We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all test scenarios are covered.

- We use two techniques - **Equivalence Partitioning & Boundary Value Analysis testing techniques** to achieve this.

1. Equivalent Class Partitioning

Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases.

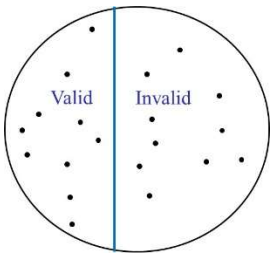
It divides the input domain of a program in to classes(valid and invalid) of data from which test cases can be derived. If the input is

Range of values	:	1 valid and 2 invalid classes
Specific value	:	1 valid and 2 invalid classes
Member of a set	:	1 valid and 1 invalid classes
Boolean	:	1 valid and 1 invalid classes

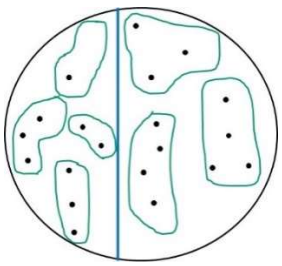
Each and every condition of particular partition (group) works as same as other. If a condition in a partition is valid, other conditions are valid too. If a condition in a partition is invalid, other conditions are invalid too.

It helps to reduce the total number of test cases from infinite to finite. The selected test cases from these groups ensure coverage of all possible scenarios.

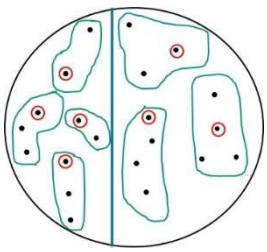
i) First-level partitioning: Valid vs. Invalid test cases



ii) Partition valid and invalid test cases into equivalence classes



iii) Create a test case for at least one value from each equivalence class



Example 1: (Range of Values)
Assume, we have to test a field which accepts Age 18 – 56

AGE Enter Age *Accepts value 18 to 56

EQUIVALENCE PARTITIONING		
Invalid	**Valid**	**Invalid**
<=17	18-56	>=57

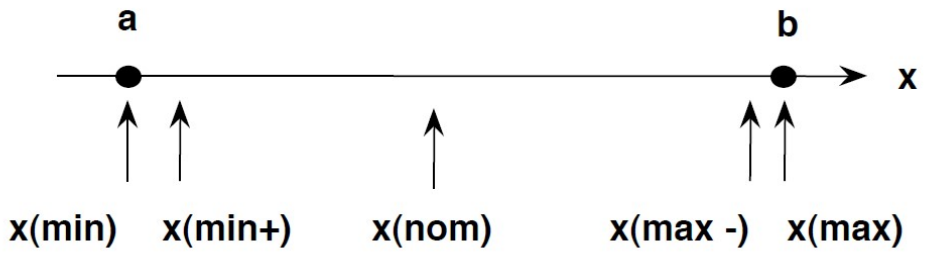
Valid Class: 18 – 56 = Pick any one input test data from 18 – 56
Invalid Class 1: <=17 = Pick any one input test data less than or equal to 17
Invalid Class 2: >=57 = Pick any one input test data greater than or equal to 57
We have one valid and two invalid conditions here.

Equivalence partitioning is not a 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.

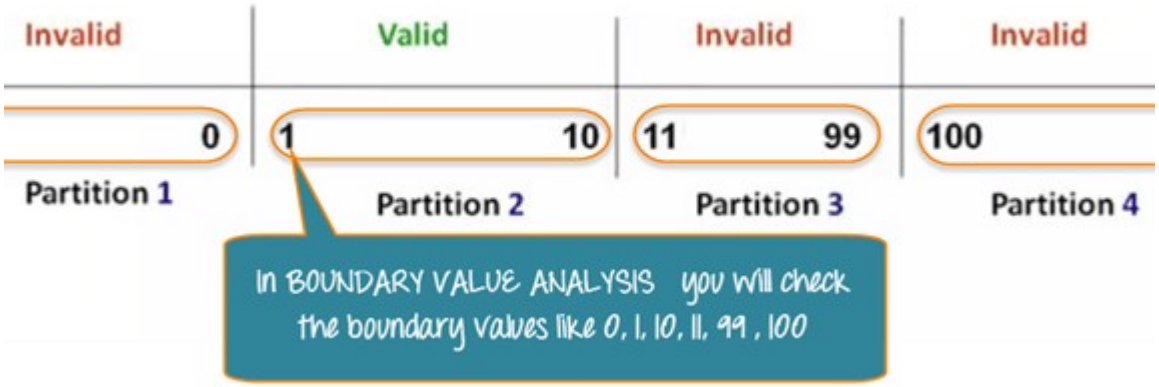
2. Boundary Value Analysis

- Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behaviour at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.
- Extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



In Boundary Value Analysis, you test boundaries between equivalence partitions



Instead of checking, one value for each partition you will check the values at the boundaries of the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at **both valid and invalid boundaries**. Boundary Value Analysis is also called **range checking**.

Assume, we have to test a field which accepts Age 18 – 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

Minimum boundary value is 18

Maximum boundary value is 56

Valid Inputs: 18,19,55,56

Invalid Inputs: 17 and 57

Why Equivalence & Boundary Analysis Testing

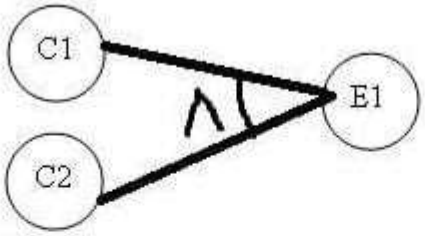
1. This testing is used to reduce a very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with a large number of variables/inputs

3. Cause Effect Graphing

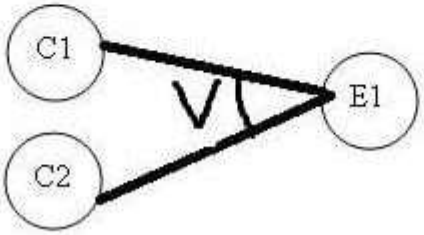
Cause and effect graph is a dynamic test case writing technique. Here causes are the input conditions and effects are the results of those input conditions.The goal is to reduce the total number of test cases still achieving the desired application quality by covering the necessary test cases for maximum coverage.

The Cause-Effect graph technique restates the requirements specification in terms of the logical relationship between the input and output conditions. Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.

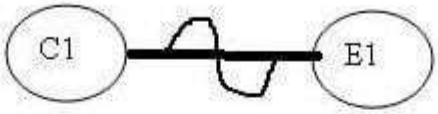
AND – For effect E1 to be true, both the causes C1 and C2 should be true



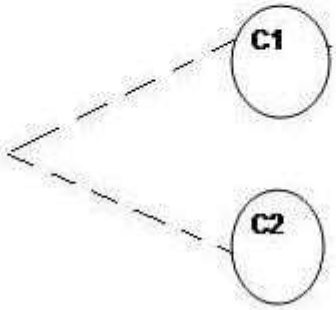
OR – For effect E1 to be true, either of causes C1 OR C2 should be true



NOT – For Effect E1 to be True, Cause C1 should be false



MUTUALLY EXCLUSIVE – When only one of the causes will hold true.



Four Steps of Cause Effect Graphing

I. Causes and Effects are listed and identifiers assigned
II. Cause Effect Graph is Developed
III. Graph converted to decision table
IV. Decision table rules converted to test cases

Example:

The “Print message” is software that read two characters and, depending on their values, messages must be printed.

- The first character must be an “A” or a “B”.
- The second character must be a digit.
- If the first character is an “A” or “B” and the second character is a digit, the file must be updated.
- If the first character is incorrect (not an “A” or “B”), the message X must be printed.
- If the second character is incorrect (not a digit), the message Y must be printed.

Solution:

Step1:

The causes for this situation are:

C1 – First character is A
C2 – First character is B
C3 – the Second character is a digit

The effects (results) for this situation are

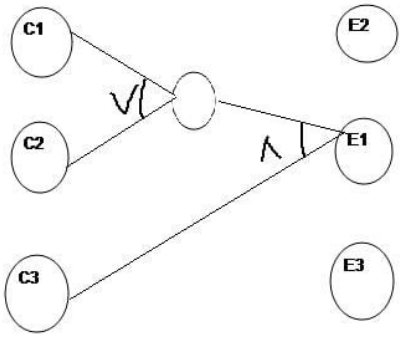
E1 – Update the file
E2 – Print message “X”
E3 – Print message “Y”

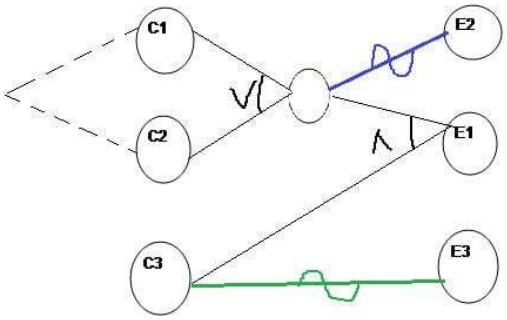
Step 2: Cause Effect Graph

In this example, let’s start with Effect E1.

Effect E1 is to update the file. The file is updated when
– The first character is “A” and the second character is a digit
– The first character is “B” and the second character is a digit
– The first character can either be “A” or “B” and cannot be both.
Now let’s put these 3 points in symbolic form:

For E1 to be true – following are the causes:
– C1 and C3 should be true
– C2 and C3 should be true
– C1 and C2 cannot be true together. This means C1 and C2 are mutually exclusive.
Now let’s draw this:





Step 3: Decision Table

Actions	TC1	TC2	TC3	TC4	TC5	TC6
C1	1	0	0	0	1	0
C2	0	1	0	0	0	1
C3	1	1	0	1	0	0
E1	1	1	0	0	0	0
E2	0	0	1	1	0	0
E3	0	0	0	0	1	1

Step4: Decision table to Test Cases