Functional (Black Box) Testing

Black Box Testing

- Program is treated as a black box
- Implementation details don't matter
- Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with excepted results.

Introduction To Functional (Black box) Testing

Figure 2.1. Functional (Black Box) testing

- Design test cases with high probability to fail
- Program is treated as a black box
- Implementation details don't matter
- Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with excepted results. Internal structure is completely ignored

What do you test in Functional Testing

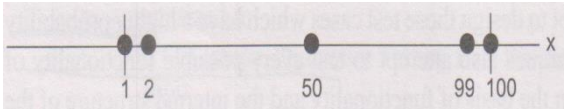
- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed

Types of Functional Testing

- Boundary Value Testing
- Equivalence Class Testing
- Decision Table based testing

- **Important Terms:**
  - **Single Fault Assumption –** We assume that fault in the program occurs due to value of just one variable.
  - **Multiple Fault Assumption -** We assume that fault in the program occurs due to value of more than one variable.

Boundary Value Analysis (BVA)



- Minimum value
- Just above minimum value
- Maximum value
- Just below maximum value
- Nominal (average) value

- In general for 'n' input variables, we get **4n + 1** test cases.

Table 2.1. Test cases for the 'Square' program

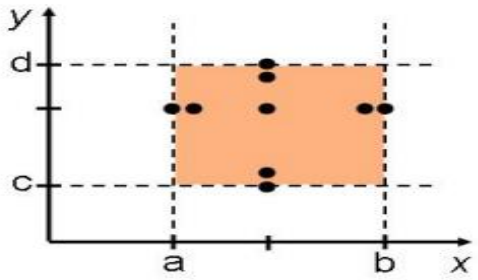
Test Case	Input x	Expected output
1.	1	1
2.	2	4
3.	50	2500
4.	99	9801
5.	100	10000

BVA Example1

- Consider a program "Addition" with two input values x and y and it gives the addition of x and y as output. The range of both input values are given as :

$$100 \leq x \leq 300$$

$$200 \leq y \leq 400$$



BVA Example1 solution

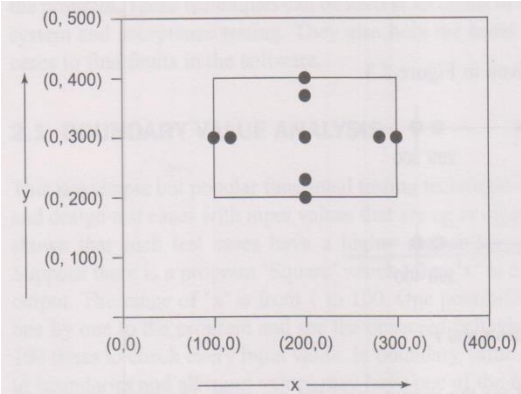


Table 2.2. Test cases for the program 'Addition'

Test Case	x	y	Expected Output
1.	100	300	400
2.	101	300	401
3.	200	300	500
4.	299	300	599
5.	300	300	600
6.	200	200	400
7.	200	201	401
8.	200	300	500
9.	200	399	599
10.	200	400	600

BVA Example 2

- Consider a program for the determination of the largest amongst three numbers . Its input values are x, y and z and values are from the interval [1, 300]. Design the boundary value test cases.

BVA Example 2 solution

Table 2.3. Boundary value test cases to find the largest among three numbers

Test Case	x	y	z	Expected output
1.	1	150	150	150
2.	2	150	150	150
3.	150	150	150	150
4.	299	150	150	299
5.	300	150	150	300
6.	150	1	150	150
7.	150	2	150	150
8.	150	299	150	299
9.	150	300	150	300
10.	150	150	1	150
11.	150	150	2	150
12.	150	150	299	299
13.	150	150	300	300

BVA Example 3

Example 2.2: Consider a program for the determination of division of a student based on the marks in three subjects. Its input is a triple of positive integers (say mark1, mark2, and mark3) and values are from interval [0, 100].
The division is calculated according to the following rules:

Marks Obtained (Average)	Division
75 – 100	First Division with distinction
60 – 74	First division
50 – 59	Second division
40 – 49	Third division
0 – 39	Fail

Total marks obtained are the average of marks obtained in the three subjects i.e.

$$\text{Average} = (\text{mark1} + \text{mark 2} + \text{mark3}) / 3$$

The program output may have one of the following words:
[Fail, Third Division, Second Division, First Division, First Division with Distinction]
Design the boundary value test cases.

BVA Example 3 solution

Table 2.4. Boundary value test cases for the program determining the division of a student

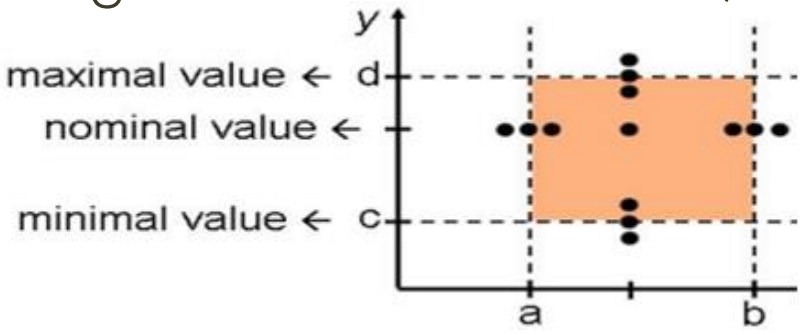
Test Case	mark1	mark2	mark3	Expected Output
1.	0	50	50	Fail
2.	1	50	50	Fail
3.	50	50	50	Second Division
4.	99	50	50	First Division
5.	100	50	50	First Division
6.	50	0	50	Fail
7.	50	1	50	Fail
8.	50	99	50	First Division
9.	50	100	50	First Division
10.	50	50	0	Fail
11.	50	50	1	Fail
12.	50	50	99	First Division
13.	50	50	100	First Division

Extensions of BVA

- Robustness testing
- Worst Case Testing
- Robust Worst Case Testing
- Special Value Testing
- Random Value Testing

Robustness Testing

- Robustness is defined as the degree to which a system operates correctly in the presence of invalid inputs.
- Robustness testing mainly focuses on exception handling i.e. how well a system responds to invalid inputs
- The robustness test cases for a function of 1 variable whose value ranges from 1 to 100 are: 0, 1, 2, 50, 99, 100, 101.
- **6n + 1** test cases



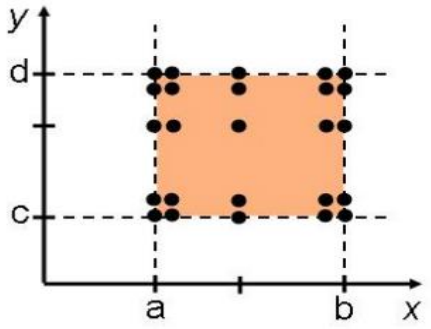
Robustness Testing Example

Table 2.7. Robustness test cases for two input values x and y

Test Case	x	y	Expected Output
1.	99	300	Invalid Input
2.	100	300	400
3.	101	300	401
4.	200	300	500
5.	299	300	599
6.	300	300	600
7.	301	300	Invalid Input
8.	200	199	Invalid Input
9.	200	200	400
10.	200	201	401
11.	200	399	599
12.	200	400	600
13.	200	401	Invalid Input

Worst case testing

- Rejects single fault assumption and tests all combinations of values.
- 5^n test cases
- Worst case testing is applicable where the physical variables have numerous interactions and where the failure of a function is extremely costly



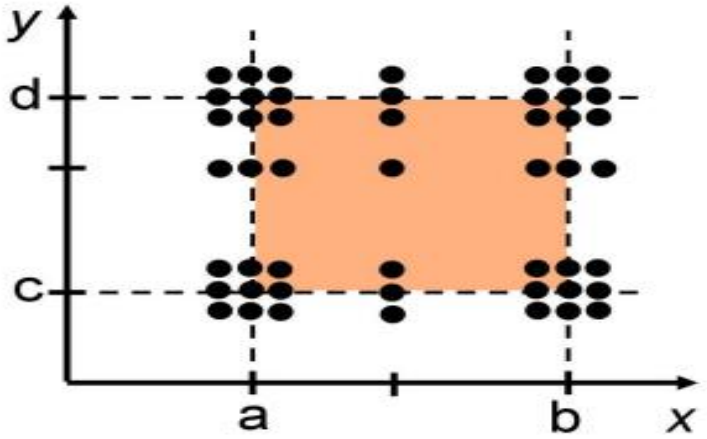
Worst case testing Example

Table 2.8. Worst test cases for the program 'Addition'

Test Case	x	y	Expected Output
1.	100	200	300
2.	100	201	301
3.	100	300	400
4.	100	399	499
5.	100	400	500
6.	101	200	301
7.	101	201	302
8.	101	300	401
9.	101	399	500
10.	101	400	501
11.	200	200	400
12.	200	201	401
13.	200	300	500
14.	200	399	599
15.	200	400	600
16.	299	200	499
17.	299	201	500
18.	299	300	599
19.	299	399	698
20.	299	400	699
21.	300	200	500
22.	300	201	501
23.	300	300	600
24.	300	399	699
25.	300	400	700

Robust Worst Case Testing

- Combination of worst case testing with robust testing.
- 7^n test cases



Special Value Testing

- Tester uses the domain knowledge and experience to test the software.
- It is an Ad-hoc testing and not based on any principle.
- It is the least systematic of all methods.
- It mainly depends on the ability of the testers.

For ex: In the nextDate function [a function where we input a date and the result is next date], a tester may use his knowledge to test for Feb 29 and leap year

Random Testing

- Use a random number generator to pick the test case value.
- Deciding how many random test cases are sufficient is a problem.

Limitations of BVA

- Bounds may not be always obvious. (Deciding upper bound for some variable may be difficult).
- Bounds may not be appropriate (Ex: Boolean values).
- No bounds could be applied in case of logic variables.