Black-box testing techniques

Durga Prasad Mohapatra
Professor
Dept. of CSE
NIT Rourkela

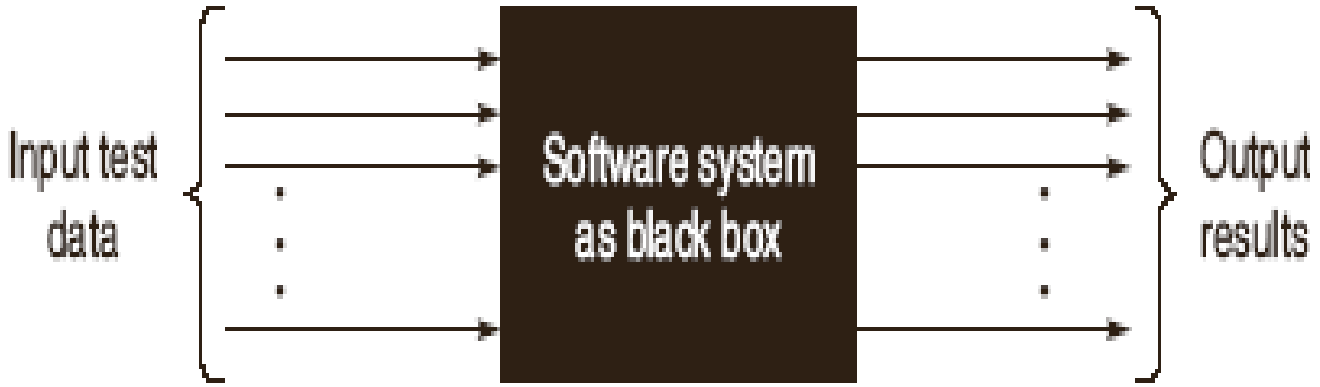
Black-box Testing

- Test cases are designed using only functional specification of the software:
  - without any knowledge of the internal structure of the software.
- For this reason, black-box testing is also known as functional testing.

Black-box Testing

- There are essentially two main approaches to design black box test cases:
  - Equivalence class partitioning
  - Boundary value analysis

Black-box Testing



Equivalence Class Partitioning

- Input values to a program are partitioned into equivalence classes.
- Partitioning is done such that:
  - program behaves in similar ways to every input value belonging to an equivalence class.

Why define equivalence classes?

- Test the code with just one representative value from each equivalence class:
  - as good as testing using any other values from the equivalence classes.

Equivalence Class Partitioning

- How do you determine the equivalence classes?
  - examine the input data.
  - few general guidelines for determining the equivalence classes can be given

Equivalence Class Partitioning

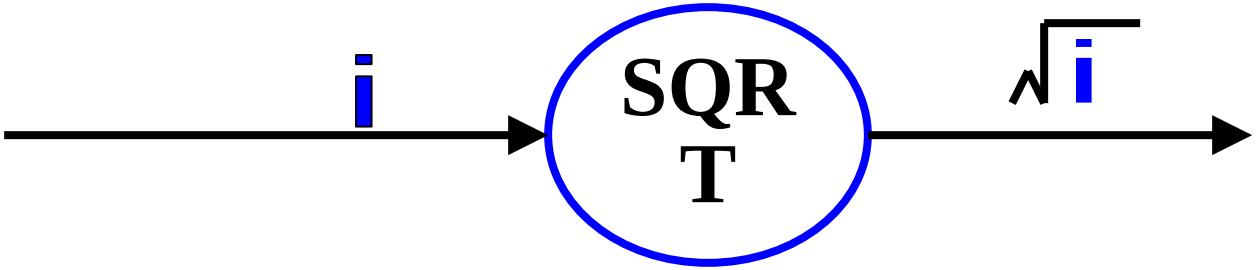
- If the input data to the program is specified by a range of values:
  - e.g. numbers between 1 to 5000.
  - one valid and two invalid equivalence classes are defined.

Equivalence Class Partitioning

- If input is an enumerated set of values:
  - e.g. {a,b,c}
  - one equivalence class for valid input values
  - another equivalence class for invalid input values should be defined.

Example

- A program reads an input value in the range of 1 and 5000:
  - computes the square root of the input number

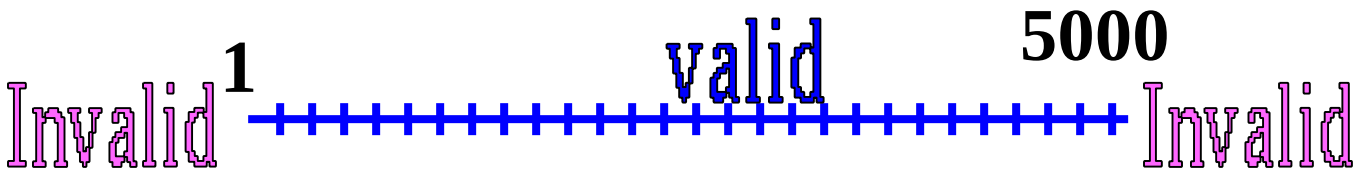


Example (cont.)

- There are three equivalence classes:
  - the set of negative integers,
  - set of integers in the range of 1 and 5000,
  - integers larger than 5000.

Example (cont.)

- The test suite must include:
  - representatives from each of the three equivalence classes:
  - a possible test suite can be:
  - {-5,500,6000}.



Example

- A program reads three numbers, A, B, and C, with a range [1, 50] and prints the largest number. Design test cases for this program using equivalence class testing technique.

Solution

- 1. First we partition the domain of input as valid input values and invalid values, getting the following classes:
- I1 = {<A, B, C> : 1 ≤ A ≤ 50}
- I2 = {<A, B, C> : 1 ≤ B ≤ 50}
- I3 = {<A, B, C> : 1 ≤ C ≤ 50}
- I4 = {<A, B, C> :A < 1}

Solution

- I5 = {<A, B, C> :A > 50}
- I6 = {<A, B, C> : B < 1}
- I7 = {<A, B, C> : B > 50}
- I8 = {<A, B, C> : C < 1}
- I9 = {<A, B, C> : C > 50}

Solution

- Now the test cases can be designed from the above derived classes, taking
- one test case from each class such that the test case covers maximum valid input classes, and
- separate test cases for each invalid class.

Solution

- The test cases are shown below:

Test case ID	A	B	C	Expected result	Classes covered by the test case
1	13	25	36	C is greatest	I_1, I_2, I_3
2	0	13	45	Invalid input	I_4
3	51	34	17	Invalid input	I_5
4	29	0	18	Invalid input	I_6
5	36	53	32	Invalid input	I_7
6	27	42	0	Invalid input	I_8
7	33	21	51	Invalid input	I_9

Solution

- 2. We can derive another set of equivalence classes based on some possibilities for three integers, A, B, and C. These are given below:
- I1 = {<A, B, C> : A > B, A > C}
- I2 = {<A, B, C> : B > A, B > C}
- I3 = {<A, B, C> : C > A, C > B}

Solution

- I4 = {<A, B, C> :A = B, A ≠ C}
- I5 = {<A, B, C> : B = C, A ≠ B}
- I6 = {<A, B, C> :A = C, C ≠ B}
- I7 = {<A, B, C> :A = B = C}

Solution

Test case ID	A	B	C	Expected Result	Classes Covered by the test case
1	25	13	13	A is greatest	I_1, I_5
2	25	40	25	B is greatest	I_2, I_6
3	24	24	37	C is greatest	I_3, I_4
4	25	25	25	All three are equal	I_7

Example

- A program determines the next date in the calendar. Its input is entered in the form of with the following range:
- $1 \leq mm \leq 12$
- $1 \leq dd \leq 31$
- $1900 \leq yyyy \leq 2025$

Example

- Its output would be the next date or an error message ‘invalid date.’ Design test cases using equivalence class partitioning method.

Solution

- First we partition the domain of input in terms of valid input values and invalid values, getting the following classes:
- I1 = { <m, d, y> : 1 ≤ m ≤ 12}
- I2 = {<m, d, y> : 1 ≤ d ≤ 31}
- I3 = {<m, d, y> : 1900 ≤ y ≤ 2025}
- I4 = {<m, d, y> : m < 1}

Solution

- I5 = {<m, d, y> : m > 12}
- I6 = {<m, d, y> : d < 1}
- I7 = {<m, d, y> : d > 31}
- I8 = {<m, d, y> : y < 1900}
- I9 = {<m, d, y> : y > 2025}

Solution

- The test cases can be designed from the above derived classes, taking one test case from each class such that the test case covers maximum valid input classes, and separate test cases for each invalid class. The test cases are shown below:

Solution

Test case ID	mm	dd	yyyy	Expected result	Classes covered by the test case
1	5	20	1996	21-5-1996	I_1, I_2, I_3
2	0	13	2000	Invalid input	I_4
3	13	13	1950	Invalid input	I_5
4	12	0	2007	Invalid input	I_6
5	6	32	1956	Invalid input	I_7
6	11	15	1899	Invalid input	I_8
7	10	19	2026	Invalid input	I_9

Example

- A program takes an angle as input within the range [0, 360] and determines in which quadrant the angle lies. Design test cases using equivalence class partitioning method.

Solution

- 1. First we partition the domain of input as valid and invalid values, getting the follow
- I1 = {<Angle> : 0 ≤ Angle ≤ 360}
- I2 = {<Angle> : Angle < 0}
- I3 = {<Angle> : Angle > 0}

Solution

- The test cases designed from these classes are shown below:

Test Case ID	Angle	Expected results	Classes covered by the test case
1	50	I Quadrant	I_1
2	–1	Invalid input	I_2
3	361	Invalid input	I_3

Solution

- 2. The classes can also be prepared based on the output criteria as shown below:
- O1 = {<Angle>: First Quadrant, if 0 ≤ Angle ≤ 90}
- O2 = {<Angle>: Second Quadrant, if 91 ≤ Angle ≤ 180}
- O3 = {<Angle>: Third Quadrant, if 181 ≤ Angle ≤ 270}

Solution

- O4 = {<Angle>: Fourth Quadrant, if 271 ≤ Angle ≤ 360}
- O5 = {<Angle>: Invalid Angle};
- However, O5 is not sufficient to cover all invalid conditions this way. Therefore, it must be further divided into equivalence classes as shown in next slide:

Solution

- $O5_1$ = {<Angle>: Invalid Angle, if Angle < 0}
- $O5_2$ = {<Angle>: Invalid Angle, if Angle > 360}

Solution

- Now the test cases can be designed from the above derived classes as shown below:

Test Case ID	Angle	Expected results	Classes covered by the test case
1	50	I Quadrant	O_1
2	135	II Quadrant	O_2
3	250	III Quadrant	O_3
4	320	IV Quadrant	O_4
5	370	Invalid angle	O_{51}
6	−1	Invalid angle	O_{52}

Boundary Value Analysis

- Some typical programming errors occur:
  - at boundaries of equivalence classes
  - might be purely due to psychological factors.
- Programmers often fail to see:
  - special processing required at the boundaries of equivalence classes.

Boundary Value Analysis

- Programmers may improperly use < instead of <=
- Boundary value analysis:
  - select test cases at the boundaries of different equivalence classes.

Example

- For a function that computes the square root of an integer in the range of 1 and 5000:
  - test cases must include the values: {0,1,5000,5001}.

Black Box testing

- Black-box testing attempts to find errors in the following categories:
- 1.To test the modules independently .
- 2.To test the functional validity of the software so that incorrect or missing functions can be recognized .
- 3.To look for interface errors.

Black Box testing

- 4. To test the system behavior and check its performance □
- 5. To test the maximum load or stress on the system.
- 6. To test the software such that the user/customer accepts the system within defined acceptable limits.

BOUNDARY VALUE ANALYSIS (BVA)

- BVA offers several methods to design test cases. Following are the few methods used:
- **1.** BOUNDARY VALUE CHECKING (BVC)
- **2.** ROBUSTNESS TESTING METHOD
- **3.** WORST-CASE TESTING METHOD
- **4.** ROBUST WORST-CASE TESTING METHOD

BOUNDARY VALUE CHECKING (BVC)

- In this method, the test cases are designed by holding one variable at its extreme value and other variables at their nominal values in the input domain.

- The variable at its extreme value can be selected at:

BOUNDARY VALUE CHECKING (BVC)

- (a) Minimum value (Min)
- (b) Value just above the minimum value (Min+)
- (c) Maximum value (Max)
- (d) Value just below the maximum value (Max−)

BOUNDARY VALUE CHECKING (BVC)

- Let us take the example of two variables, A and B.
- If we consider all the above combinations with nominal values, then following test cases (see Fig. 1) can be designed:

- 1. Anom, Bmin
- 2. Anom, Bmin+
- 3. Anom, Bmax
- 4. Anom, Bmax–
- 5. Amin, Bnom
- 6. Amin+, Bnom
- 7. Amax, Bnom
- 8. Amax–, Bnom
- 9. Anom, Bnom

BOUNDARY VALUE CHECKING (BVC)

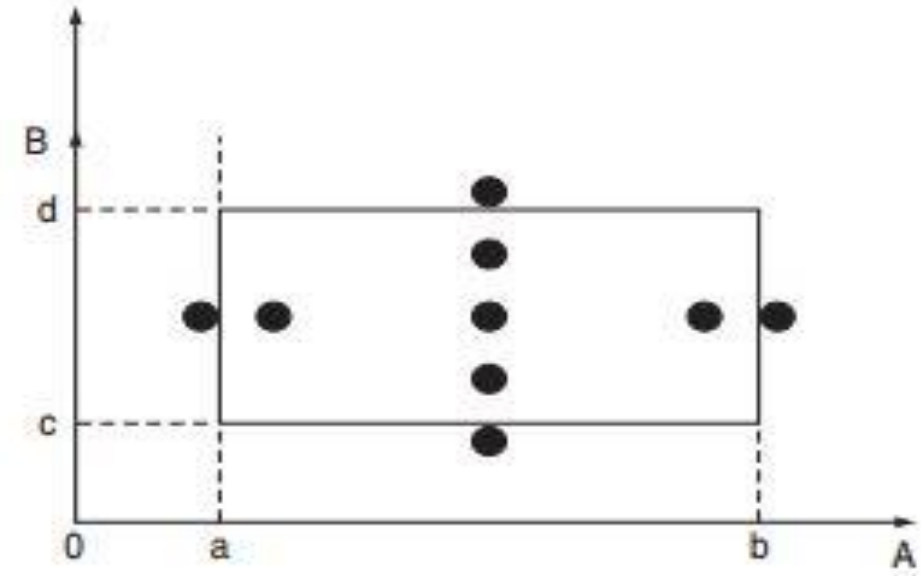


Fig 1: Boundary Value Checking

BOUNDARY VALUE CHECKING (BVC)

- It can be generalized that for n variables in a module, 4n + 1 test cases can be designed with boundary value checking method.

ROBUSTNESS TESTING METHOD

- The idea of BVC can be extended such that boundary values are exceeded as: □
- 1. A value just greater than the Maximum value (Max+)
- 2. A value just less than Minimum value (Min−)

ROBUSTNESS TESTING METHOD

- When test cases are designed considering the above points in addition to BVC, it is called robustness testing.

- Let us take the previous example again. Add the following test cases to the list of 9 test cases designed in BVC:

- 10. Amax+, Bnom 11. Amin–, Bnom
- 12. Anom, Bmax+ 13. Anom, Bmin–

ROBUSTNESS TESTING METHOD

- It can be generalized that for n input variables in a module, 6n + 1 test cases can be designed with robustness testing.

WORST-CASE TESTING METHOD

- We can again extend the concept of BVC by assuming more than one variable on the boundary.
- It is called worst-case testing method.
- Again, take the previous example of two variables, A and B. We can add the following test cases to the list of 9 test cases designed in BVC as:

WORST-CASE TESTING METHOD

10. Amin, Bmin
11. Amin+, Bmin
12. Amin, Bmin+
13. Amin+, Bmin+
14. Amax, Bmin
15. Amax–, Bmin
16. Amax, Bmin+
17. Amax–, Bmin+
18. Amin, Bmax
19. Amin+, Bmax
20. Amin, Bmax–
21. Amin+,Bmax–
22. Amax, Bmax
23. Amax–, Bmax
24. Amax, Bmax–
25. Amax–,Bmax–

WORST-CASE TESTING METHOD

- It can be generalized that for n input variables in a module, 5^n test cases can be designed with worst-case testing.

ROBUST WORST-CASE TESTING METHOD

- In the previous method, the extreme values of a variable considered are of BVC only.
- The worst case can be further extended if we consider robustness also, that is,
- in worst case testing if we consider the extreme values of the variables as in robustness testing method covered in Robustness Testing

ROBUST WORST-CASE TESTING METHOD

- Again take the example of two variables, A and B. We can add the following test cases to the list of 25 test cases designed in previous section.
- 26. Amin-, Bmin-
- 27. Amin-, Bmin
- 28. Amin, Bmin-
- 29. Amin-, Bmin+
- 30. Amin+, Bmin-
- 31. Amin-, Bmax

- 32. Amax, Bmin-
- 33. Amin-, Bmax-
- 34. Amax-, Bmin-
- 35. Amax+,Bmax+
- 36. Amax+, Bmin
- 37. Amin, Bmin+
- 38. Amax+, Bmin+
- 39. Amax+,Bmax+
- 40. Amax+,Bmax
- 41. Amax, Bmax+
- 42. Amax+,Bmax-
- 43. Amax-, Bmax+
- 44. Amax+,Bnom
- 45. Anom, Bmax+
- 46. Amin-,Bnom
- 47. Anom, Bmin-
- 48. Amax+,Bmin-
- 49. Amin-, Bmax+

Example

A program reads an integer number within the range [1,100] and determines whether it is a prime number or not. Design test cases for this program using BVC, robust testing, and worst-case testing methods.

Test cases using BVC

- Since there is one variable, the total number of
- test cases will be 4n + 1 = 5.
- In our example, the set of minimum and maximum values is shown below:

- Min value = 1
- Min+ value = 2
- Max value = 100
- Max– value = 99
- Nominal value = 50–55

- Using these values, test cases can be designed as shown below:

Test Case ID	Integer Variable	Expected Output
1	1	Not a prime number
2	2	Prime number
3	100	Not a prime number
4	99	Not a prime number
5	53	Prime number

Test cases using robust testing

- Since there is one variable, the total number of test cases will be 6n + 1 = 7. The set of boundary values is shown below:

- Min value = 1
- Min– value = 0
- Min+ value = 2
- Max value = 100
- Max− value = 99
- Max+ value = 101
- Nominal value = 50–55

- Using these values, test cases can be designed as shown below:

Test Case ID	Integer Variable	Expected Output
1	0	Invalid input
2	1	Not a prime number
3	2	Prime number
4	100	Not a prime number
5	99	Not a prime number
6	101	Invalid input
7	53	Prime number

Test cases using worst-case testing

- Since there is one variable, the total number of test cases will be $5^n = 5$.

- Therefore, the number of test cases will be same as BVC.

Example

- A program computes a^b where a lies in the range [1,10] and b within [1,5].

- Design test cases for this program using BVC, robust testing, and worst-case testing methods.

Test cases using BVC

- Since there are two variables, a and b, the total number of test cases will be 4n + 1 = 9. The set of boundary values is shown below:

	a	b
Min value	1	1
Min+ value	2	2
Max value	10	5
Max− value	9	4
Nominal value	5	3

Using these values, test cases can be designed as shown below:

Test Case ID	a	b	Expected Output
1	1	3	1
2	2	3	8
3	10	3	1000
4	9	3	729
5	5	1	5
6	5	2	25
7	5	4	625
8	5	5	3125
9	5	3	125

Test cases using robust testing

- Since there are two variables, a and b, the total number of test cases will be 6n + 1 = 13.
- The set of boundary values is shown below:

	a	b
Min value	1	1
Min– value	0	0
Min+ value	2	2
Max value	10	5
Max+ value	11	6
Max– value	9	4
Nominal value	5	3

Using these values, test cases can be designed as shown below:

Test Case ID	a	b	Expected output
1	0	3	Invalid input
2	1	3	1
3	2	3	8
4	10	3	1000
5	11	3	Invalid input
6	9	3	729
7	5	0	Invalid input
8	5	1	5
9	5	2	25
10	5	4	625
11	5	5	3125

Test cases using worst-case testing

- Since there are two variables, a and b, the total number of test cases will be $5^n = 25$.
- The set of boundary values is shown below:

	a	b
Min value	1	1
Min+ value	2	2
Max value	10	5
Max− value	9	4
Nominal value	5	3

There may be more than one variable at extreme values in this case. Therefore, test cases can be designed as shown below :

Test Case ID	a	b	Expected Output
1	1	1	1
2	1	2	1
3	1	3	3
4	1	4	1
5	1	5	1
6	2	1	2
7	2	2	4
8	2	3	8
9	2	4	16
10	2	5	32
11	5	1	5
12	5	2	25
13	5	3	125
14	5	4	625
15	5	5	3125
16	9	1	9
17	9	2	81
18	9	3	729
19	9	4	6561
20	9	5	59049
21	10	1	10
22	10	2	100
23	10	3	1000
24	10	4	10000
25	10	5	100000