



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**.

Errors detected by 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. ☐

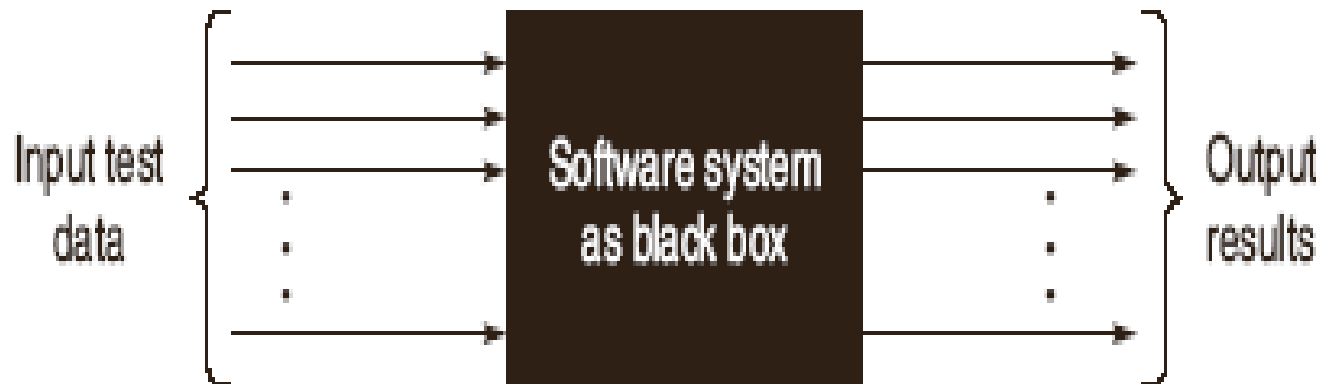
Errors detected by black box testing

- 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.

Black-box Testing Techniques

- There are many approaches to design black box test cases:
 - Equivalence class partitioning
 - Boundary value analysis
 - State table based testing
 - Decision table based testing
 - Cause-effect graph based testing
 - Orthogonal array testing
 - Positive-negative testing

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.

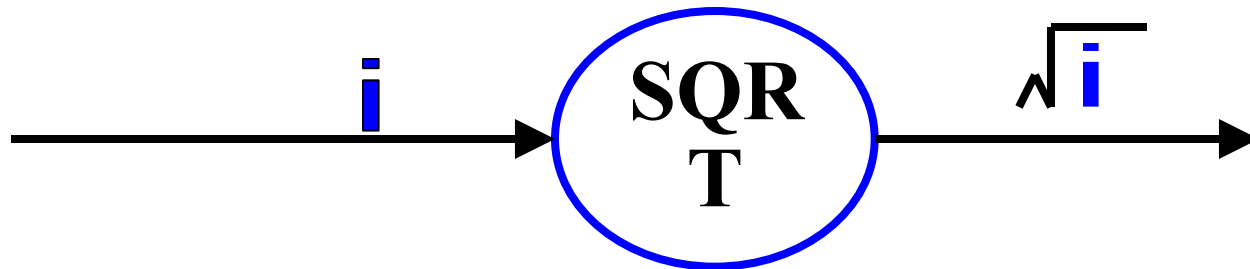


Equivalence Class Partitioning

- 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 - I

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



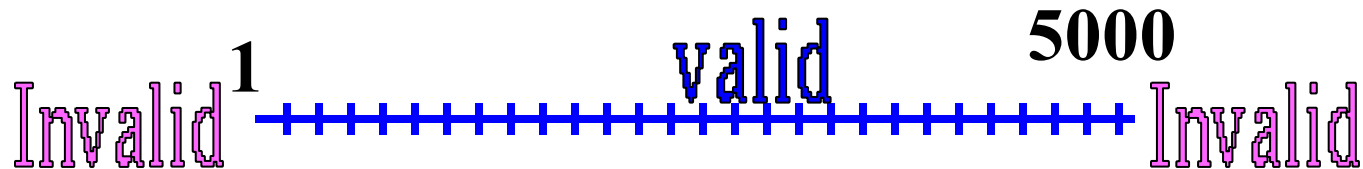
Example - I (cont.)

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



Example - I (cont.)

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



Example - 2

- 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

I. First we partition the domain of input as valid input values and invalid values, getting the following classes:

- $I_1 = \{ \langle A, B, C \rangle : 1 \leq A \leq 50 \}$
- $I_2 = \{ \langle A, B, C \rangle : 1 \leq B \leq 50 \}$
- $I_3 = \{ \langle A, B, C \rangle : 1 \leq C \leq 50 \}$
- $I_4 = \{ \langle A, B, C \rangle : A < 1 \}$

Solution

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

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 | l_1, l_2, l_3 |
| 2 | 0 | 13 | 45 | Invalid input | l_4 |
| 3 | 51 | 34 | 17 | Invalid input | l_5 |
| 4 | 29 | 0 | 18 | Invalid input | l_6 |
| 5 | 36 | 53 | 32 | Invalid input | l_7 |
| 6 | 27 | 42 | 0 | Invalid input | l_8 |
| 7 | 33 | 21 | 51 | Invalid input | l_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:

- $I_1 = \{ \langle A, B, C \rangle : A > B, A > C \}$
- $I_2 = \{ \langle A, B, C \rangle : B > A, B > C \}$
- $I_3 = \{ \langle A, B, C \rangle : C > A, C > B \}$

Solution

- $I_4 = \{ \langle A, B, C \rangle : A = B, A \neq C \}$
- $I_5 = \{ \langle A, B, C \rangle : B = C, A \neq B \}$
- $I_6 = \{ \langle A, B, C \rangle : A = C, C \neq B \}$
- $I_7 = \{ \langle A, B, C \rangle : A = B = C \}$

Solution

| Test case ID | A | B | C | Expected Result | Classes Covered by the test case |
|--------------|----|----|----|---------------------|----------------------------------|
| 1 | 25 | 13 | 13 | A is greatest | l_1, l_5 |
| 2 | 25 | 40 | 25 | B is greatest | l_2, l_6 |
| 3 | 24 | 24 | 37 | C is greatest | l_3, l_4 |
| 4 | 25 | 25 | 25 | All three are equal | l_7 |

Example - 3

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:

- $I_1 = \{ \langle m, d, y \rangle : 1 \leq m \leq 12 \}$
- $I_2 = \{ \langle m, d, y \rangle : 1 \leq d \leq 31 \}$
- $I_3 = \{ \langle m, d, y \rangle : 1900 \leq y \leq 2025 \}$
- $I_4 = \{ \langle m, d, y \rangle : m < 1 \}$

Solution

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

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 in next slide.

Solution

| Test case ID | mm | dd | yyyy | Expected result | Classes covered by the test case |
|--------------|----|----|------|-----------------|----------------------------------|
| 1 | 5 | 20 | 1996 | 21-5-1996 | l_1, l_2, l_3 |
| 2 | 0 | 13 | 2000 | Invalid input | l_4 |
| 3 | 13 | 13 | 1950 | Invalid input | l_5 |
| 4 | 12 | 0 | 2007 | Invalid input | l_6 |
| 5 | 6 | 32 | 1956 | Invalid input | l_7 |
| 6 | 11 | 15 | 1899 | Invalid input | l_8 |
| 7 | 10 | 19 | 2026 | Invalid input | l_9 |

Example - 4

- 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

I. First, we partition the domain of input as valid and invalid classes as follows:

- $I_1 = \{ \langle \text{Angle} \rangle : 0 \leq \text{Angle} \leq 360 \}$
- $I_2 = \{ \langle \text{Angle} \rangle : \text{Angle} < 0 \}$
- $I_3 = \{ \langle \text{Angle} \rangle : \text{Angle} > 360 \}$

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 | l_1 |
| 2 | -1 | Invalid input | l_2 |
| 3 | 361 | Invalid input | l_3 |

Solution

2. The classes can also be prepared based on the output criteria as shown below:

- $O1 = \{ \langle \text{Angle} \rangle : \text{First Quadrant, if } 0 \leq \text{Angle} \leq 90 \}$
- $O2 = \{ \langle \text{Angle} \rangle : \text{Second Quadrant, if } 91 \leq \text{Angle} \leq 180 \}$
- $O3 = \{ \langle \text{Angle} \rangle : \text{Third Quadrant, if } 181 \leq \text{Angle} \leq 270 \}$

Solution

- $O4 = \{ \langle \text{Angle} \rangle : \text{Fourth Quadrant, if } 271 \leq \text{Angle} \leq 360 \}$
- $O5 = \{ \langle \text{Angle} \rangle : \text{Invalid Angle} \};$
- 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

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

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} |

Summary

- Discussed the errors detected by black-box test testing techniques.
- Explained equivalence partitioning technique with some examples.

References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 4), Second Edition, Oxford University Press, 2016.



Thank You