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

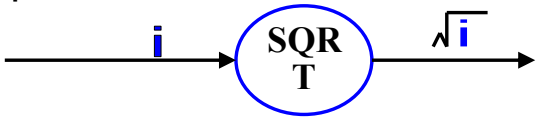
- If the input data to the program is specified by a range of values:
  - e.g. numbers between 1 to 5000.
  - one valid and two invalid equivalence classes are defined.

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 - 1

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



Example - 1 (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 - 1 (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

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 - 3

A program determines the next date in the calendar. Its input is entered in the form of with the following range:

- 1 ≤ mm ≤ 12
- 1 ≤ dd ≤ 31
- 1900 ≤ yyyy ≤ 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 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	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 - 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

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

- I1 = {<Angle> : 0 ≤ Angle ≤ 360}
- I2 = {<Angle> :Angle < 0}
- I3 = {<Angle> :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	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 > 360}
- O5$_2$={<Angle>: 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