Software Testing

Alpha, Beta and Acceptance Testing

The term **Acceptance Testing** is used when the software is developed for a specific customer. A series of tests are conducted to enable the customer to validate all requirements. These tests are conducted by the end user / customer.

The terms alpha and beta testing are used when the software is developed as a product for anonymous customers.

Alpha Tests are conducted at the developer's site by some potential customers. These tests are conducted in a controlled environment. Alpha testing may be started when formal testing process is near completion.

Beta Tests are conducted by the customers / end users at their sites. Unlike alpha testing, developer is not present here. Beta testing is conducted in a real environment that cannot be controlled by the developer.

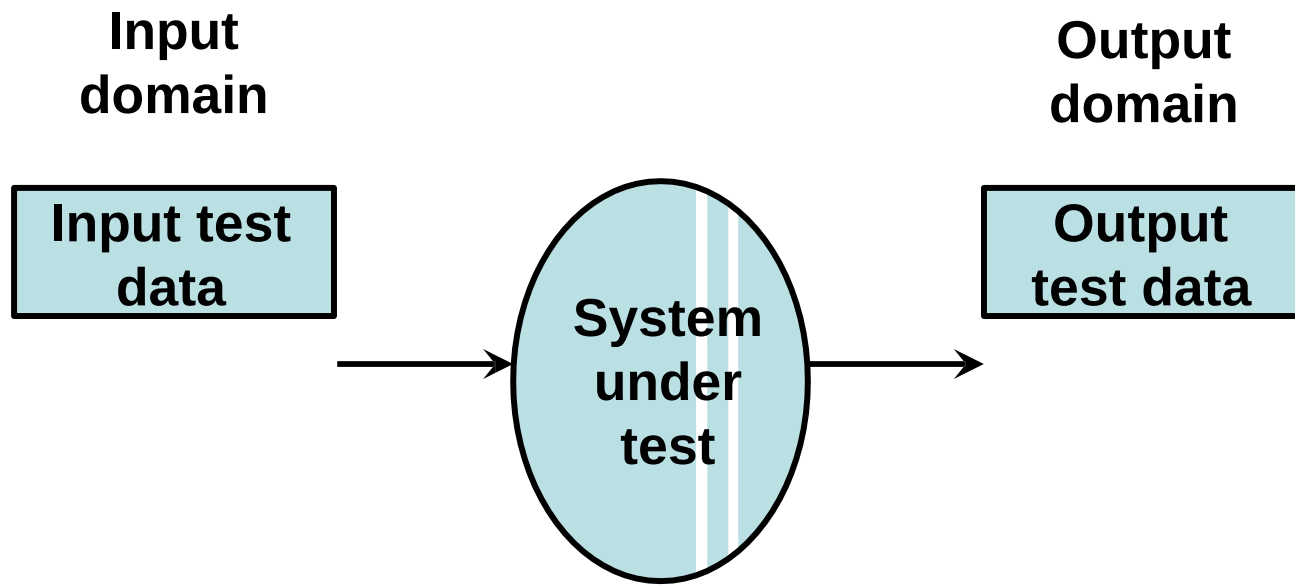# *Software Testing*

Functional Testing

Input domain

Output domain

Input test data

System under test

Output test data

Fig. 3: Black box testing

Boundary Value Analysis

Consider a program with two input variables x and y. These input variables have specified boundaries as:
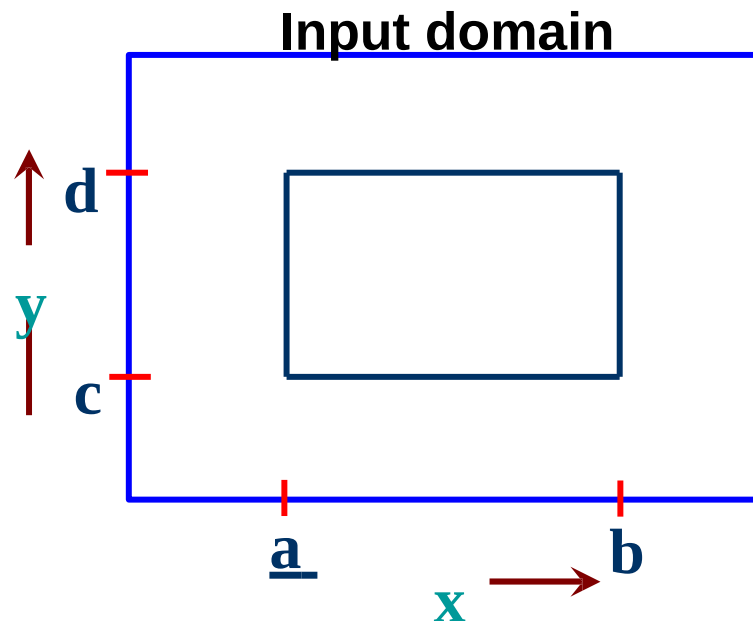
$$a \leq x \leq b$$

$$c \leq y \leq d$$

Input domain

Fig.4: Input domain for program having two input variables

Software Testing

The boundary value analysis test cases for our program with two inputs variables (x and y) that may have any value from 100 to 300 are: (200,100), (200,101), (200,200), (200,299), (200,300), (100,200), (101,200), (299,200) and
(300,200). This input domain is shown in Fig. 8.5. Each dot represent a test case and inner rectangle is the domain of legitimate inputs. Thus, for a program of n variables, boundary value analysis yield **4n + 1** test cases.
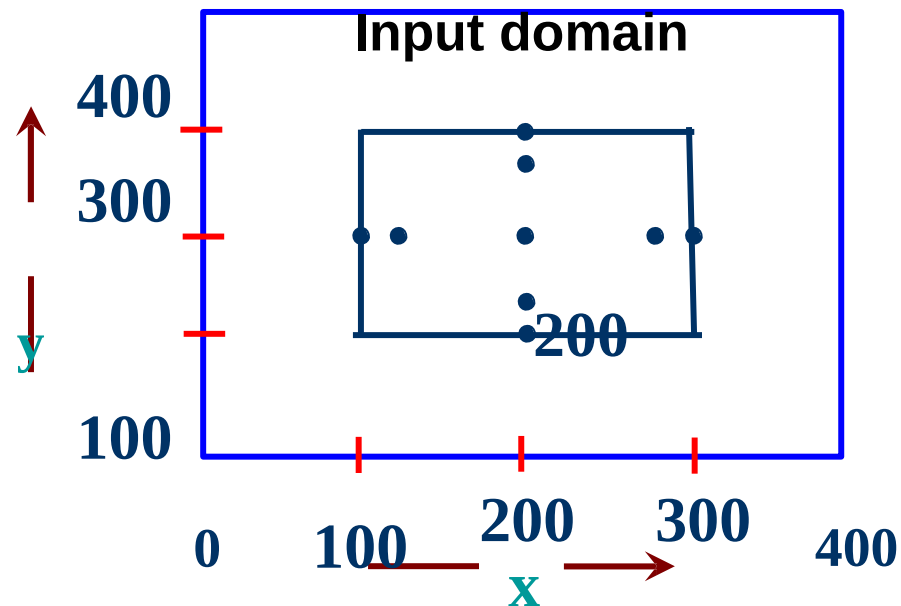
Fig. 5: Input domain of two variables x and y with boundaries [100,300] each

15

Software Testing

Example 8.1

Consider a program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a,b,c) and values may be from interval [0,100]. The program output may have one of the following words.

[**Not a quadratic equation**; **Real roots**; **Imaginary roots**; **Equal roots**]

Design the boundary value test cases.

Software Testing

```c
#include <conio.h>
#include <math.h>
1       int main()
2    {
3       int  a,b,c,validInput=0,d;
4       double D;
5       printf("Enter the 'a' value: ");
6       scanf("%d",&a);
7       printf("Enter the 'b' value: ");
8       scanf("%d",&b);
9       printf("Enter the 'c' value: ");
10      scanf("%d",&c);
11      if ((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c >= 0)
           && (c <= 100)) {
12        validInput = 1;
13        if (a == 0) {
14           validInput = -1;
15        }
16      }
17      if (validInput==1) {
18        d = b*b - 4*a*c;
19        if (d == 0) {
20           printf("The roots are equal and are r1 = r2 = %f\n",
                   -b/(2*(float) a));
```

```
21        }
22      else if ( d > 0 ) {
23          D=sqrt(d);
24          printf("The roots are real and are r1 = %f and r2 = %f\n",
                    (-b-D)/(2* a),  (-b+D)/(2* a));
25        }
26      else {
27          D=sqrt(-d)/(2*a);
28          printf("The roots are imaginary and are r1 = (%f,%f) and
                    r2 = (%f,%f)\n", -b/(2.0*a),D,-b/(2.0*a),-D);
29        }
30    }
31    else if (validInput == -1) {
32        printf("The vlaues do not constitute a Quadratic equation.");
33    }
34    else {
35        printf("The inputs belong to invalid range.");
36    }
37    getche();
38    return 1;
39  }
```

Fig. 19: Code of quadratic equation problem

Software Testing

Solution

Quadratic equation will be of type:

$$ax^2+bx+c=0$$

Roots are real if $(b^2-4ac)>0$

Roots are imaginary if $(b^2-4ac)<0$

Roots are equal if $(b^2-4ac)=0$

Equation is not quadratic if $a=0$

Software Testing

The boundary value test cases are :

Test Case	a	b	c	Expected output
1	0	50	50	Not Quadratic
2	1	50	50	Real Roots
3	50	50	50	Imaginary Roots
4	99	50	50	Imaginary Roots
5	100	50	50	Imaginary Roots
6	50	0	50	Imaginary Roots
7	50	1	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Equal Roots
10	50	50	0	Real Roots
11	50	50	1	Real Roots
12	50	50	99	Imaginary Roots
13	50	50	100	Imaginary Roots

Software Testing

Example 8.2

Consider a program for determining the Previous date. Its input is a triple of day, month and year with the values in the range

$1 \leq$ month ≤ 12

$1 \leq$ day $\leq \quad 31$

$1900 \leq$ year ≤ 2025

The possible outputs would be Previous date or invalid input date. Design the boundary value test cases.

Software

```
    /* Program to generate the previous date given a date, assumes data
given as dd mm yyyy separated by space and performs error checks on the
validity of the current date entered. */

    #include <stdio.h>
    #include <conio.h>

1    int main()
2    {
3       int day, month, year, validDate = 0;
     /*Date Entry*/
4       printf("Enter the day value: ");
5       scanf("%d", &day);
6       printf("Enter the month value: ");
7       scanf("%d", &month);
8       printf("Enter the year value: ");
9       scanf("%d",&year);
     /*Check Date Validity */
10      if (year >= 1900 && year <= 2025) {
11        if (month == 1 || month == 3 || month == 5 || month == 7 ||
            month == 8 || month == 10 || month == 12) {
```

(Contd.)...

```
12          if (day >= 1 && day <= 31) {
13              validDate = 1;
14          }
15          else {
16              validDate = 0;
17          }
18      }
19      else if (month == 2) {
20          int rVal=0;
21          if (year%4 == 0) {
22              rVal=1;
23              if ((year%100)==0 && (year % 400) !=0) {
24                  rVal=0;
25              }
26          }
27          if (rVal ==1 && (day >=1 && day <=29) ) {
28              validDate = 1;
29          }
30          else if (day >=1 && day <= 28 ) {
31              validDate = 1;
32          }
```

```
33          else {
34              validDate = 0;
35          }
36      }
37      else if ((month >= 1 && month <= 12)  && (day >= 1 && day <= 30)) {
38          validDate = 1;
39      }
40      else {
41          validDate = 0;
42      }
43   }
     /*Prev Date Calculation*/
44   if (validDate) {
45      if (day == 1) {
46          if (month == 1) {
47              year--;
48              day=31;
49              month=12;
50          }
51          else if (month == 3) {
52              int rVal=0;
```

(Contd.)...

Software Testing

```
53          if (year%4 == 0) {
54             rVal=1;
55             if ((year%100)==0 && (year % 400) !=0) {
56          rVal=0;
57       }
58          }
59          if (rVal ==1) {
60             day=29;
61             month--;
62          }
63          else {
64             day=28;
65             month--;
66          }
67       }
68       else if (month == 2 || month == 4 || month == 6 || month == 9 ||
         month == 11) {
69          day = 31;
70          month--;
```

```
71              }
72            else {
73               day=30;
74               month--;
75            }
76         }
77       else {
78         day--;
79       }
80       printf("The next date is: %d-%d-%d",day,month,year);
81     }
82     else {
83       printf("The entered date ( %d-%d-%d ) is invalid",day,month, year);
84     }
85     getche ();
86     return 1;
87   }
```

Fig. 15: Program for previous date problem

Software Testing

Solution

The Previous date program takes a date as input and checks it for validity. If valid, it returns the previous date as its output.

With single fault assumption theory, 4n+1 test cases can be designed and which are equal to 13.

Software Testing

The boundary value test cases are:

Test Case	Month	Day	Year	Expected output
1	6	15	1900	14 June, 1900
2	6	15	1901	14 June, 1901
3	6	15	1962	14 June, 1962
4	6	15	2024	14 June, 2024
5	6	15	2025	14 June, 2025
6	6	1	1962	31 May, 1962
7	6	2	1962	1 June, 1962
8	6	30	1962	29 June, 1962
9	6	31	1962	Invalid date
10	1	15	1962	14 January, 1962
11	2	15	1962	14 February, 1962
12	11	15	1962	14 November, 1962
13	12	15	1962	14 December, 1962

<u>Example 8.3</u>

Consider a simple program to classify a triangle. Its inputs is a triple of positive integers (say x, y, z) and the data type for input parameters ensures that these will be integers greater than 0 and less than or equal to 100.

The program output may be one of the following words:

[**<u>Scalene</u>**; **<u>Isosceles</u>**; **<u>Equilateral; Not a triangle</u>**]
Design the boundary value test cases.

Software

```c
#include <stdio.h>
#include <conio.h>
1    int main()
2    {
3       int a,b,c,validInput=0;
4       printf("Enter the side 'a' value: ");
5       scanf("%d",&a);
6       printf("Enter the side 'b' value: ")
7       scanf("%d",&b);
8       printf("Enter the side 'c' value:");
9       scanf("%d",&c);
10      if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c > 0)
            && (c <= 100)) {
11       if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
12          validInput = 1;
13       }
14      }
15      else {
16         validInput = -1;
17      }
18      If (validInput==1) {
19         If ((a==b) && (b==c)) {
20            printf("The trinagle is equilateral");
21         }
22         else if ( (a == b) || (b == c) || (c == a) ) {
```

```
23            printf("The triangle is isosceles");
24         }
25      else {
26         printf("The trinagle is scalene");
27      }
28   }
29   else if (validInput == 0) {
30      printf("The values do not constitute a Triangle");
31   }
32   else {
33      printf("The inputs belong to invalid range");
34   }
35   getche();
36   return 1;
37 }
```

Fig. 20 : Code of triangle classification problem

Solution

The boundary value test cases are shown below:

Test case	x	y	z	Expected Output
1	50	50	1	Isosceles
2	50	50	2	Isosceles
3	50	50	50	Equilateral
4	50	50	99	Isosceles
5	50	50	100	Not a triangle
6	50	1	50	Isosceles
7	50	2	50	Isosceles
8	50	99	50	Isosceles
9	50	100	50	Not a triangle
10	1	50	50	Isosceles
11	2	50	50	Isosceles
12	99	50	50	Isosceles
13	100	50	50	Not a triangle

Software Testing

It is nothing but the extension of boundary value analysis. Here, we would like to see, what happens when the extreme values are exceeded with a value slightly greater than the maximum, and a value slightly less than minimum. It means, we want to go outside the legitimate boundary of input domain.

This extended form of boundary value analysis is called robustness testing and shown in Fig. 6

There are four additional test cases which are outside the legitimate input domain. Hence total test cases in robustness testing are **6n+1,** where n is the number of input variables. So, 13 test cases are:

(200,99), (200,100), (200,101), (200,200), (200,299), (200,300)

(200,301), (99,200), (100,200), (101,200), (299,200), (300,200), (301,200)

Software Testing
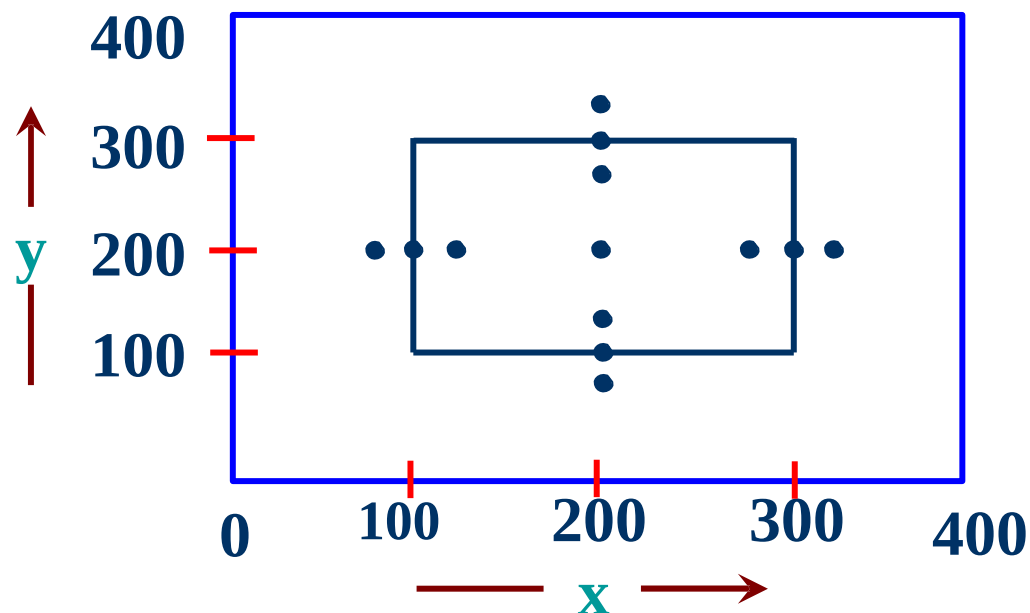
Fig. 8.6: Robustness test cases for two variables x and y with range [100,300] each

Software Testing

Worst-case testing

If we like to see what happens when more than one variable has an extreme value. In electronic circuits analysis, this is called "worst case analysis". It is more thorough in the sense that boundary value test cases are a proper subset of worst case test cases. It requires more effort. Worst case testing for a function of n variables generate 5^n test cases as opposed to **4n+1** test cases for boundary value analysis. Our two variables example will have **5^2=25** test cases and are given in table 1.

Software Testing

Table 1: Worst cases test inputs for two variables example

Test case number	Inputs		Test case number	Inputs	
	x	*y*		*x*	*y*
1	100	100	14	**200**	**299**
2	100	101	15	**200**	**300**
3	**100**	**200**	16	299	100
4	100	299	17	299	101
5	100	300	18	**299**	**200**
6	101	100	19	299	299
7	101	101	20	299	300
8	**101**	**200**	21	300	100
9	101	299	22	300	101
10	101	300	23	**300**	**200**
11	**200**	**100**	24	300	299
12	**200**	**101**	25	300	300
13	**200**	**200**	--		

Software Testing

Example 8.4

Consider the program for the determination of nature of roots of a quadratic equation as explained in example 8.1. Design the Robust test case and worst test cases for this program.

Software Testing

<u>Solution</u>

Robust test cases are 6n+1. Hence, in 3 variable input cases total number of test cases are 19 as given on next slide:

Software

Test case	a	b	c	Expected Output
1	-1	50	50	Invalid input`
2	0	50	50	Not quadratic equation
3	1	50	50	Real roots
4	50	50	50	Imaginary roots
5	99	50	50	Imaginary roots
6	100	50	50	Imaginary roots
7	101	50	50	Invalid input
8	50	-1	50	Invalid input
9	50	0	50	Imaginary roots
10	50	1	50	Imaginary roots
11	50	99	50	Imaginary roots
12	50	100	50	Equal roots
13	50	101	50	Invalid input
14	50	50	-1	Invalid input
15	50	50	0	Real roots
16	50	50	1	Real roots
17	50	50	99	Imaginary roots
18	50	50	100	Imaginary roots
19	50	50	101	Invalid input

Software Testing

In case of worst test case total test cases are 5^n. Hence, **125** test cases will be generated in worst test cases. The worst test cases are given below:

Test Case	a	b	c	Expected output
1	0	0	0	Not Quadratic
2	0	0	1	Not Quadratic
3	0	0	50	Not Quadratic
4	0	0	99	Not Quadratic
5	0	0	100	Not Quadratic
6	0	1	0	Not Quadratic
7	0	1	1	Not Quadratic
8	0	1	50	Not Quadratic
9	0	1	99	Not Quadratic
10	0	1	100	Not Quadratic
11	0	50	0	Not Quadratic
12	0	50	1	Not Quadratic
13	0	50	50	Not Quadratic
14	0	50	99	Not Quadratic

Software Testing

Test Case	A	b	c	Expected output
15	0	50	100	Not Quadratic
16	0	99	0	Not Quadratic
17	0	99	1	Not Quadratic
18	0	99	50	Not Quadratic
19	0	99	99	Not Quadratic
20	0	99	100	Not Quadratic
21	0	100	0	Not Quadratic
22	0	100	1	Not Quadratic
23	0	100	50	Not Quadratic
24	0	100	99	Not Quadratic
25	0	100	100	Not Quadratic
26	1	0	0	Equal Roots
27	1	0	1	Imaginary
28	1	0	50	Imaginary
29	1	0	99	Imaginary
30	1	0	100	Imaginary
31	1	1	0	Real Roots

Software Testing

Test Case	A	b	C	Expected output
32	1	1	1	Imaginary
33	1	1	50	Imaginary
34	1	1	99	Imaginary
35	1	1	100	Imaginary
36	1	50	0	Real Roots
37	1	50	1	Real Roots
38	1	50	50	Real Roots
39	1	50	99	Real Roots
40	1	50	100	Real Roots
41	1	99	0	Real Roots
42	1	99	1	Real Roots
43	1	99	50	Real Roots
44`	1	99	99	Real Roots
45	1	99	100	Real Roots
46	1	100	0	Real Roots
47	1	100	1	Real Roots
48	1	100	50	Real Roots

Software Testing

Test Case	A	b	C	Expected output
49	1	100	99	Real Roots
50	1	100	100	Real Roots
51	50	0	0	Equal Roots
52	50	0	1	Imaginary
53	50	0	50	Imaginary
54	50	0	99	Imaginary
55	50	0	100	Imaginary
56	50	1	0	Real Roots
57	50	1	1	Imaginary
58	50	1	50	Imaginary
59	50	1	99	Imaginary
60	50	1	100	Imaginary
61	50	50	0	Real Roots
62	50	50	1	Real Roots
63	50	50	50	Imaginary
64	50	50	99	Imaginary
65	50	50	100	Imaginary

Software Testing

Test Case	A	b	C	Expected output
66	50	99	0	Real Roots
67	50	99	1	Real Roots
68	50	99	50	Imaginary
69	50	99	99	Imaginary
70	50	99	100	Imaginary
71	50	100	0	Real Roots
72	50	100	1	Real Roots
73	50	100	50	Equal Roots
74	50	100	99	Imaginary
75	50	100	100	Imaginary
76	99	0	0	Equal Roots
77	99	0	1	Imaginary
78	99	0	50	Imaginary
79	99	0	99	Imaginary
80	99	0	100	Imaginary
81	99	1	0	Real Roots
82	99	1	1	Imaginary

Software

Test Case	A	b	C	Expected output
83	99	1	50	Imaginary
84	99	1	99	Imaginary
85	99	1	100	Imaginary
86	99	50	0	Real Roots
87	99	50	1	Real Roots
88	99	50	50	Imaginary
89	99	50	99	Imaginary
90	99	50	100	Imaginary
91	99	99	0	Real Roots
92	99	99	1	Real Roots
93	99	99	50	Imaginary Roots
94	99	99	99	Imaginary
95	99	99	100	Imaginary
96	99	100	0	Real Roots
97	99	100	1	Real Roots
98	99	100	50	Imaginary
99	99	100	99	Imaginary
100	99	100	100	Imaginary

Software Testing

Test Case	A	b	C	Expected output
101	100	0	0	Equal Roots
102	100	0	1	Imaginary
103	100	0	50	Imaginary
104	100	0	99	Imaginary
105	100	0	100	Imaginary
106	100	1	0	Real Roots
107	100	1	1	Imaginary
108	100	1	50	Imaginary
109	100	1	99	Imaginary
110	100	1	100	Imaginary
111	100	50	0	Real Roots
112	100	50	1	Real Roots
113	100	50	50	Imaginary
114	100	50	99	Imaginary
115	100	50	100	Imaginary
116	100	99	0	Real Roots
117	100	99	1	Real Roots
118	100	99	50	Imaginary

Software Testing

Test Case	A	b	C	Expected output
119	**100**	**99**	**99**	Imaginary
120	**100**	**99**	**100**	Imaginary
121	**100**	**100**	**0**	Real Roots
122	**100**	**100**	**1**	Real Roots
123	**100**	**100**	**50**	Imaginary
124	**100**	**100**	**99**	Imaginary
125	**100**	**100**	**100**	Imaginary

Software Testing

Example 8.5

Consider the program for the determination of previous date in a calendar as explained in example 8.2. Design the robust and worst test cases for this program.

Software Testing

Solution

Robust test cases are 6n+1. Hence total 19 robust test cases are designed and are given on next slide.

Software

Test case	Month	Day	Year	Expected Output
1	6	15	1899	Invalid date (outside range)
2	6	15	1900	14 June, 1900
3	6	15	1901	14 June, 1901
4	6	15	1962	14 June, 1962
5	6	15	2024	14 June, 2024
6	6	15	2025	14 June, 2025
7	6	15	2026	Invalid date (outside range)
8	6	0	1962	Invalid date
9	6	1	1962	31 May, 1962
10	6	2	1962	1 June, 1962
11	6	30	1962	29 June, 1962
12	6	31	1962	Invalid date
13	6	32	1962	Invalid date
14	0	15	1962	Invalid date
15	1	15	1962	14 January, 1962
16	2	15	1962	14 February, 1962
17	11	15	1962	14 November, 1962
18	12	15	1962	14 December, 1962
19	13	15	1962	Invalid date

Software Testing

In case of worst test case total test cases are 5^n. Hence, 125 test cases will be generated in worst test cases. The worst test cases are given below:

Test Case	Month	Day	Year	Expected output
1	1	1	1900	31 December, 1899
2	1	1	1901	31 December, 1900
3	1	1	1962	31 December, 1961
4	1	1	2024	31 December, 2023
5	1	1	2025	31 December, 2024
6	1	2	1900	1 January, 1900
7	1	2	1901	1 January, 1901
8	1	2	1962	1 January, 1962
9	1	2	2024	1 January, 2024
10	1	2	2025	1 January, 2025
11	1	15	1900	14 January, 1900
12	1	15	1901	14 January, 1901
13	1	15	1962	14 January, 1962
14	1	15	2024	14 January, 2024

Software Testing

Test Case	A	b	c	Expected output
15	1	15	2025	14 January, 2025
16	1	30	1900	29 January, 1900
17	1	30	1901	29 January, 1901
18	1	30	1962	29 January, 1962
19	1	30	2024	29 January, 2024
20	1	30	2025	29 January, 2025
21	1	31	1900	30 January, 1900
22	1	31	1901	30 January, 1901
23	1	31	1962	30 January, 1962
24	1	31	2024	30 January, 2024
25	1	31	2025	30 January, 2025
26	2	1	1900	31 January, 1900
27	2	1	1901	31 January, 1901
28	2	1	1962	31 January, 1962
29	2	1	2024	31 January, 2024
30	2	1	2025	31 January, 2025
31	2	2	1900	1 February, 1900

Software Testing

Test Case	Month	Day	Year	Expected output
32	2	2	1901	1 February, 1901
33	2	2	1962	1 February, 1962
34	2	2	2024	1 February, 2024
35	2	2	2025	1 February, 2025
36	2	15	1900	14 February, 1900
37	2	15	1901	14 February, 1901
38	2	15	1962	14 February, 1962
39	2	15	2024	14 February, 2024
40	2	15	2025	14 February, 2025
41	2	30	1900	Invalid date
42	2	30	1901	Invalid date
43	2	30	1962	Invalid date
44	2	30	2024	Invalid date
45	2	30	2025	Invalid date
46	2	31	1900	Invalid date
47	2	31	1901	Invalid date
48	2	31	1962	Invalid date

Software Testing

Test Case	Month	Day	Year	Expected output
49	2	31	2024	Invalid date
50	2	31	2025	Invalid date
51	6	1	1900	31 May, 1900
52	6	1	1901	31 May, 1901
53	6	1	1962	31 May, 1962
54	6	1	2024	31 May, 2024
55	6	1	2025	31 May, 2025
56	6	2	1900	1 June, 1900
57	6	2	1901	1 June, 1901
58	6	2	1962	1 June, 1962
59	6	2	2024	1 June, 2024
60	6	2	2025	1 June, 2025
61	6	15	1900	14 June, 1900
62	6	15	1901	14 June, 1901
63	6	15	1962	14 June, 1962
64	6	15	2024	14 June, 2024
65	6	15	2025	14 June, 2025

Software Testing

Test Case	Month	Day	Year	Expected output
66	6	30	1900	29 June, 1900
67	6	30	1901	29 June, 1901
68	6	30	1962	29 June, 1962
69	6	30	2024	29 June, 2024
70	6	30	2025	29 June, 2025
71	6	31	1900	Invalid date
72	6	31	1901	Invalid date
73	6	31	1962	Invalid date
74	6	31	2024	Invalid date
75	6	31	2025	Invalid date
76	11	1	1900	31 October, 1900
77	11	1	1901	31 October, 1901
78	11	1	1962	31 October, 1962
79	11	1	2024	31 October, 2024
80	11	1	2025	31 October, 2025
81	11	2	1900	1 November, 1900
82	11	2	1901	1 November, 1901

Software

Test Case	Month	Day	Year	Expected output
83	11	2	1962	1 November, 1962
84	11	2	2024	1 November, 2024
85	11	2	2025	1 November, 2025
86	11	15	1900	14 November, 1900
87	11	15	1901	14 November, 1901
88	11	15	1962	14 November, 1962
89	11	15	2024	14 November, 2024
90	11	15	2025	14 November, 2025
91	11	30	1900	29 November, 1900
92	11	30	1901	29 November, 1901
93	11	30	1962	29 November, 1962
94	11	30	2024	29 November, 2024
95	11	30	2025	29 November, 2025
96	11	31	1900	Invalid date
97	11	31	1901	Invalid date
98	11	31	1962	Invalid date
99	11	31	2024	Invalid date
100	11	31	2025	Invalid date

Software Testing

Test Case	Month	Day	Year	Expected output
101	12	1	1900	30 November, 1900
102	12	1	1901	30 November, 1901
103	12	1	1962	30 November, 1962
104	12	1	2024	30 November, 2024
105	12	1	2025	30 November, 2025
106	12	2	1900	1 December, 1900
107	12	2	1901	1 December, 1901
108	12	2	1962	1 December, 1962
109	12	2	2024	1 December, 2024
110	12	2	2025	1 December, 2025
111	12	15	1900	14 December, 1900
112	12	15	1901	14 December, 1901
113	12	15	1962	14 December, 1962
114	12	15	2024	14 December, 2024
115	12	15	2025	14 December, 2025
116	12	30	1900	29 December, 1900
117	12	30	1901	29 December, 1901
118	12	30	1962	29 December, 1962

Software Testing

Test Case	Month	Day	Year	Expected output
119	12	30	2024	29 December, 2024
120	12	30	2025	29 December, 2025
121	12	31	1900	30 December, 1900
122	12	31	1901	30 December, 1901
123	12	31	1962	30 December, 1962
124	12	31	2024	30 December, 2024
125	12	31	2025	30 December, 2025

Example 8.6

Consider the triangle problem as given in example 8.3. Generate robust and worst test cases for this problem.

Software Testing

Solution

Robust test cases are given on next slide.

Software

Test Case`	x	y	z	Expected Output
1	50	50	0	**Invalid input`**
2	50	50	1	Isosceles
3	50	50	2	Isosceles
4	50	50	50	Equilateral
5	50	50	99	Isosceles
6	50	50	100	Not a triangle
7	50	50	101	**Invalid input**
8	50	0	50	**Invalid input**
9	50	1	50	Isosceles
10	50	2	50	Isosceles
11	50	99	50	Isosceles
12	50	100	50	Not a triangle
13	50	101	50	**Invalid input**
14	0	50	50	**Invalid input**
15	1	50	50	Isosceles
16	2	50	50	Isosceles
17	99	50	50	Isosceles
18	100	50	50	Not a triangle
19	101	50	50	**Invalid input**

Software Testing

Worst test cases are 125 and are given below:

Test Case	*x*	*y*	*z*	*Expected output*
1	1	1	1	Equilateral
2	1	1	2	Not a triangle
3	1	1	50	Not a triangle
4	1	1	99	Not a triangle
5	1	1	100	Not a triangle
6	1	2	1	Not a triangle
7	1	2	2	Isosceles
8	1	2	50	Not a triangle
9	1	2	99	Not a triangle
10	1	2	100	Not a triangle
11	1	50	1	Not a triangle
12	1	50	2	Not a triangle
13	1	50	50	Isosceles
14	1	50	99	Not a triangle

Software Testing

Test Case	A	b	c	Expected output
15	1	50	100	Not a triangle
16	1	99	1	Not a triangle
17	1	99	2	Not a triangle
18	1	99	50	Not a triangle
19	1	99	99	Isosceles
20	1	99	100	Not a triangle
21	1	100	1	Not a triangle
22	1	100	2	Not a triangle
23	1	100	50	Not a triangle
24	1	100	99	Not a triangle
25	1	100	100	Isosceles
26	2	1	1	Not a triangle
27	2	1	2	Isosceles
28	2	1	50	Not a triangle
29	2	1	99	Not a triangle
30	2	1	100	Not a triangle
31	2	2	1	Isosceles

Software Testing

Test Case	A	b	C	Expected output
32	2	2	2	Equilateral
33	2	2	50	Not a triangle
34	2	2	99	Not a triangle
35	2	2	100	Not a triangle
36	2	50	1	Not a triangle
37	2	50	2	Not a triangle
38	2	50	50	Isosceles
39	2	50	99	Not a triangle
40	2	50	100	Not a triangle
41	2	99	1	Not a triangle
42	2	99	2	Not a triangle
43	2	99	50	Not a triangle
44	2	99	99	Isosceles
45	2	99	100	Scalene
46	2	100	1	Not a triangle
47	2	100	2	Not a triangle
48	2	100	50	Not a triangle

Software Testing

Test Case	A	b	C	Expected output
49	2	100	50	Scalene
50	2	100	99	Isosceles
51	50	1	100	Not a triangle
52	50	1	1	Not a triangle
53	50	1	2	Isosceles
54	50	1	50	Not a triangle
55	50	1	99	Not a triangle
56	50	2	100	Not a triangle
57	50	2	1	Not a triangle
58	50	2	2	Isosceles
59	50	2	50	Not a triangle
60	50	2	99	Not a triangle
61	50	50	100	Isosceles
62	50	50	1	Isosceles
63	50	50	2	Equilateral
64	50	50	50	Isosceles
65	50	50	99	Not a triangle

Software Testing

Test Case	A	B	C	Expected output
66	50	99	1	Not a triangle
67	50	99	2	Not a triangle
68	50	99	50	Isosceles
69	50	99	99	Isosceles
70	50	99	100	Scalene
71	50	100	1	Not a triangle
72	50	100	2	Not a triangle
73	50	100	50	Not a triangle
74	50	100	99	Scalene
75	50	100	100	Isosceles
76	50	1	1	Not a triangle
77	99	1	2	Not a triangle
78	99	1	50	Not a triangle
79	99	1	99	Isosceles
80	99	1	100	Not a triangle
81	99	2	1	Not a triangle
82	99	2	2	Not a triangle

Software

Test Case	A	b	C	Expected output
83	99	2	50	Not a triangle
84	99	2	99	Isosceles
85	99	2	100	Scalene
86	99	50	1	Not a triangle
87	99	50	2	Not a triangle
88	99	50	50	Isosceles
89	99	50	99	Isosceles
90	99	50	100	Scalene
91	99	99	1	Isosceles
92	99	99	2	Isosceles
93	99	99	50	Isosceles
94	99	99	99	Equilateral
95	99	99	100	Isosceles
96	99	100	1	Not a triangle
97	99	100	2	Scalene
98	99	100	50	Scalene
99	99	100	99	Isosceles
100	99	100	100	Isosceles

Software

Test Case	A	b	C	Expected output
101	100	1	1	Not a triangle
102	100	1	2	Not a triangle
103	100	1	50	Not a triangle
104	100	1	99	Not a triangle
105	100	1	100	Isosceles
106	100	2	1	Not a triangle
107	100	2	2	Not a triangle
108	100	2	50	Not a triangle
109	100	2	99	Scalene
110	100	2	100	Isosceles
111	100	50	1	Not a triangle
112	100	50	2	Not a triangle
113	100	50	50	Not a triangle
114	100	50	99	Scalene
115	100	50	100	Isosceles
116	100	99	1	Not a triangle
117	100	99	2	Scalene
118	100	99	50	Scalene

Software Testing

Test Case	A	b	C	Expected output
119	100	99	99	Isosceles
120	100	99	100	Isosceles
121	100	100	1	Isosceles
122	100	100	2	Isosceles
123	100	100	50	Isosceles
124	100	100	99	Isosceles
125	100	100	100	Equilateral

Equivalence Class Testing

Two steps are required to implementing this method:

1. The equivalence classes are identified by **taking each input condition and partitioning it into valid and invalid classes**. For example, **if an input condition specifies a range of values from 1 to 999**, we identify **one valid equivalence class [1<item<999**]; **and two invalid equivalence classes [item<1] and [item>999].**

2. Generate the test cases using the equivalence classes identified in the previous step. This is performed by **writing test cases covering all the valid equivalence classes. Then a test case is written for each invalid equivalence class so that no test contains more than one invalid class**. This is to ensure that no two invalid classes mask each other.

Software Testing
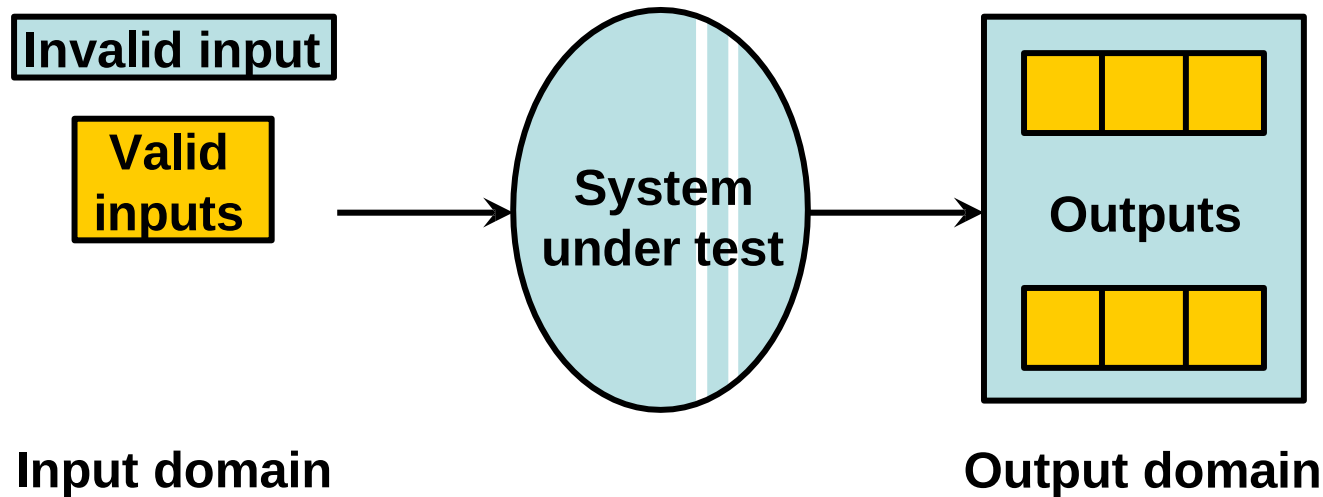
Fig. 7: Equivalence partitioning

Most of the time, equivalence class testing defines classes of the input domain. However, equivalence classes should also be defined for output domain. Hence, we should design equivalence classes based on input and output domain.

Example 8.7

Consider the program for the determination of nature of roots of a quadratic equation as explained in example 8.1. Identify the equivalence class test cases for output and input domains.

Software Testing

Output domain equivalence class test cases can be identified as follows:

O_1={<a,b,c>:Not a quadratic equation if a = 0}

O_1={<a,b,c>:Real roots if $(b^2-4ac)>0$}

O_1={<a,b,c>:Imaginary roots if $(b^2-4ac)<0$}

O_1={<a,b,c>:Equal roots if $(b^2-4ac)=0$}`

The number of test cases can be derived form above relations and shown below:

Test case	a	b	c	Expected output
1	0	50	50	Not a quadratic equation
2	1	50	50	Real roots
3	50	50	50	Imaginary roots
4	50	100	50	Equal roots

Software Testing

We may have another set of test cases based on input domain.

$I_1 = \{a: a = 0\}$

$I_2 = \{a: a < 0\}$

$I_3 = \{a: 1 \le a \le 100\}$

$I_4 = \{a: a > 100\}$

$I_5 = \{b: 0 \le b \le 100\}$

$I_6 = \{b: b < 0\}$

$I_7 = \{b: b > 100\}$

$I_8 = \{c: 0 \le c \le 100\}$

$I_9 = \{c: c < 0\}$

$I_{10} = \{c: c > 100\}$

Software Testing

Test Case	a	b	c	Expected output
1	0	50	50	Not a quadratic equation
2	-1	50	50	Invalid input
3	50	50	50	Imaginary Roots
4	101	50	50	invalid input
5	50	50	50	Imaginary Roots
6	50	-1	50	invalid input
7	50	101	50	invalid input
8	50	50	50	Imaginary Roots
9	50	50	-1	invalid input
10	50	50	101	invalid input

Here test cases 5 and 8 are redundant test cases. If we choose any value other than nominal, we may not have redundant test cases.

Hence total test cases are 10+4=**14** for this problem.

Example 8.8

Consider the program for determining the previous date in a calendar as explained in example 8.3. Identify the equivalence class test cases for output & input domains.

Software Testing

Solution

Output domain equivalence class are:

O_1={<D,M,Y>: Previous date if all are valid inputs}

O_1={<D,M,Y>: Invalid date if any input makes the date invalid}

Test case	M	D	Y	Expected output
1	6	15	1962	14 June, 1962
2	6	31	1962	Invalid date

Software Testing

We may have another set of test cases which are based on input domain.

I_1={month: $1 \leq m \leq 12$}

I_2={month: $m < 1$}

I_3={month: $m > 12$}

I_4={day: $1 \leq D \leq 31$}

I_5={day: $D < 1$}

I_6={day: $D > 31$}

I_7={year: $1900 \leq Y \leq 2025$}

I_8={year: $Y < 1900$}

I_9={year: $Y > 2025$}

Software Testing

Inputs domain test cases are :

Test Case	M	D	Y	Expected output
1	6	15	1962	14 June, 1962
2	-1	15	1962	Invalid input
3	13	15	1962	invalid input
4	6	15	1962	14 June, 1962
5	6	-1	1962	invalid input
6	6	32	1962	invalid input
7	6	15	1962	14 June, 1962
8	6	15	1899	invalid input (Value out of range)
9	6	15	2026	invalid input (Value out of range)

Example – 8.9

Consider the triangle problem specified in example 8.3. Identify the equivalence class test cases for output and input domain.

Software Testing

Solution

Output domain equivalence classes are:

$O_1 = \{<x,y,z>$: Equilateral triangle with sides $x,y,z\}$

$O_1 = \{<x,y,z>$: Isosceles triangle with sides $x,y,z\}$

$O_1 = \{<x,y,z>$: Scalene triangle with sides $x,y,z\}$

$O_1 = \{<x,y,z>$: Not a triangle with sides $x,y,z\}$

The test cases are:

Test case	*x*	*y*	*z*	*Expected Output*
1	50	50	50	Equilateral
2	50	50	99	Isosceles
3	100	99	50	Scalene
4	50	100	50	Not a triangle

Software Testing

Input domain based classes are:

$I_1 = \{x: x < 1\}$

$I_2 = \{x: x > 100\}$

$I_3 = \{x: 1 \le x \le 100\}$

$I_4 = \{y: y < 1\}$

$I_5 = \{y: y > 100\}$

$I_6 = \{y: 1 \le y \le 100\}$

$I_7 = \{z: z < 1\}$

$I_8 = \{z: z > 100\}$

$I_9 = \{z: 1 \le z \le 100\}$

Software Testing

Some inputs domain test cases can be obtained using the relationship amongst x,y and z.

$I_{10} = \{< x,y,z >: x = y = z\}$

$I_{11} = \{< x,y,z >: x = y, x \neq z\}$

$I_{12} = \{< x,y,z >: x = z, x \neq y\}$

$I_{13} = \{< x,y,z >: y = z, x \neq y\}$

$I_{14} = \{< x,y,z >: x \neq y, x \neq z, y \neq z\}$ $I_{15} = \{< x,y,z >: x = y + z\}$

$I_{16} = \{< x,y,z >: x > y + z\}$

$I_{17} = \{< x,y,z >: y = x + z\}$

$I_{18} = \{< x,y,z >: y > x + z\}$

$I_{19} = \{< x,y,z >: z = x + y\}$

$I_{20} = \{< x,y,z >: z > x + y\}$

Test cases derived from input domain are:

Test case	x	y	z	Expected Output
1	0	50	50	Invalid input
2	101	50	50	Invalid input
3	50	50	50	Equilateral
4	50	0	50	Invalid input
5	50	101	50	Invalid input
6	50	50	50	Equilateral
7	50	50	0	Invalid input
8	50	50	101	Invalid input
9	50	50	50	Equilateral
10	60	60	60	Equilateral
11	50	50	60	Isosceles
12	50	60	50	Isosceles
13	60	50	50	Isosceles

Software Testing

Test case	x	y	z	Expected Output
14	100	99	50	Scalene
15	100	50	50	Not a triangle
16	100	50	25	Not a triangle
17	50	100	50	Not a triangle
18	50	100	25	Not a triangle
19	50	50	100	Not a triangle
20	25	50	100	Not a triangle