SOFTWARE TESTING

Black-Box Testing Techniques

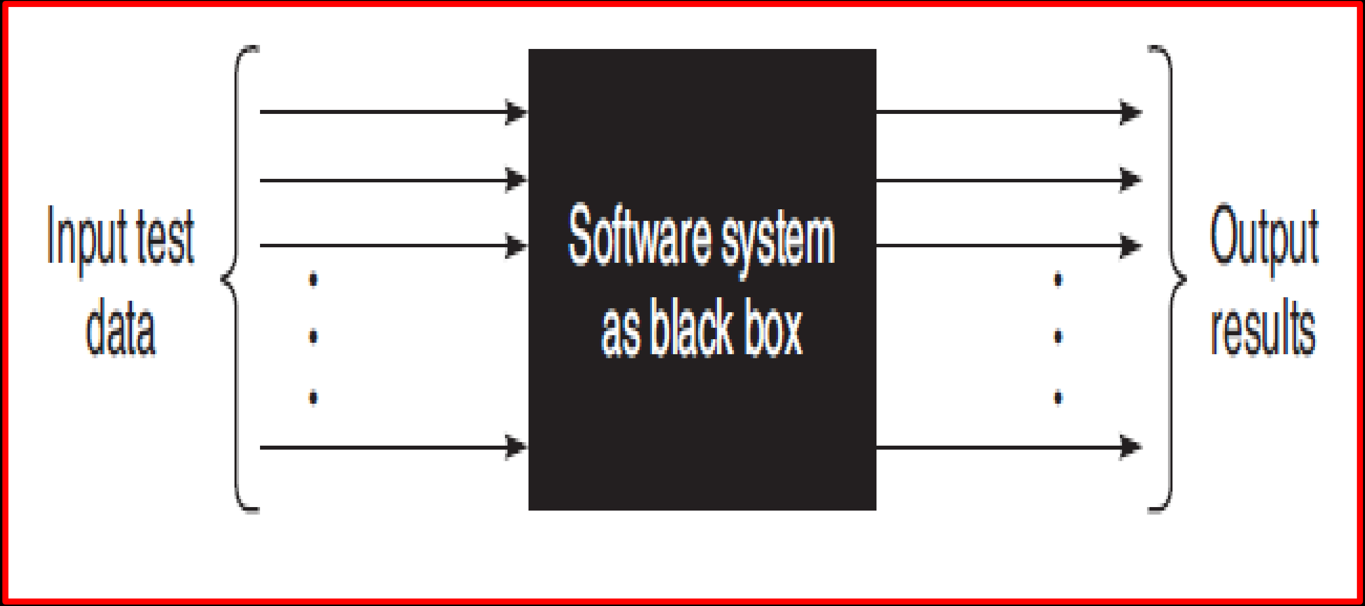
Dr. ARIVUSELVAN.K

Associate Professor – (SCORE)

VIT University

Dynamic Testing: Black-Box Testing

In black-box technique, Input test data is given to the system, which is a black box to the tester, and results are checked against expected outputs after executing the software.



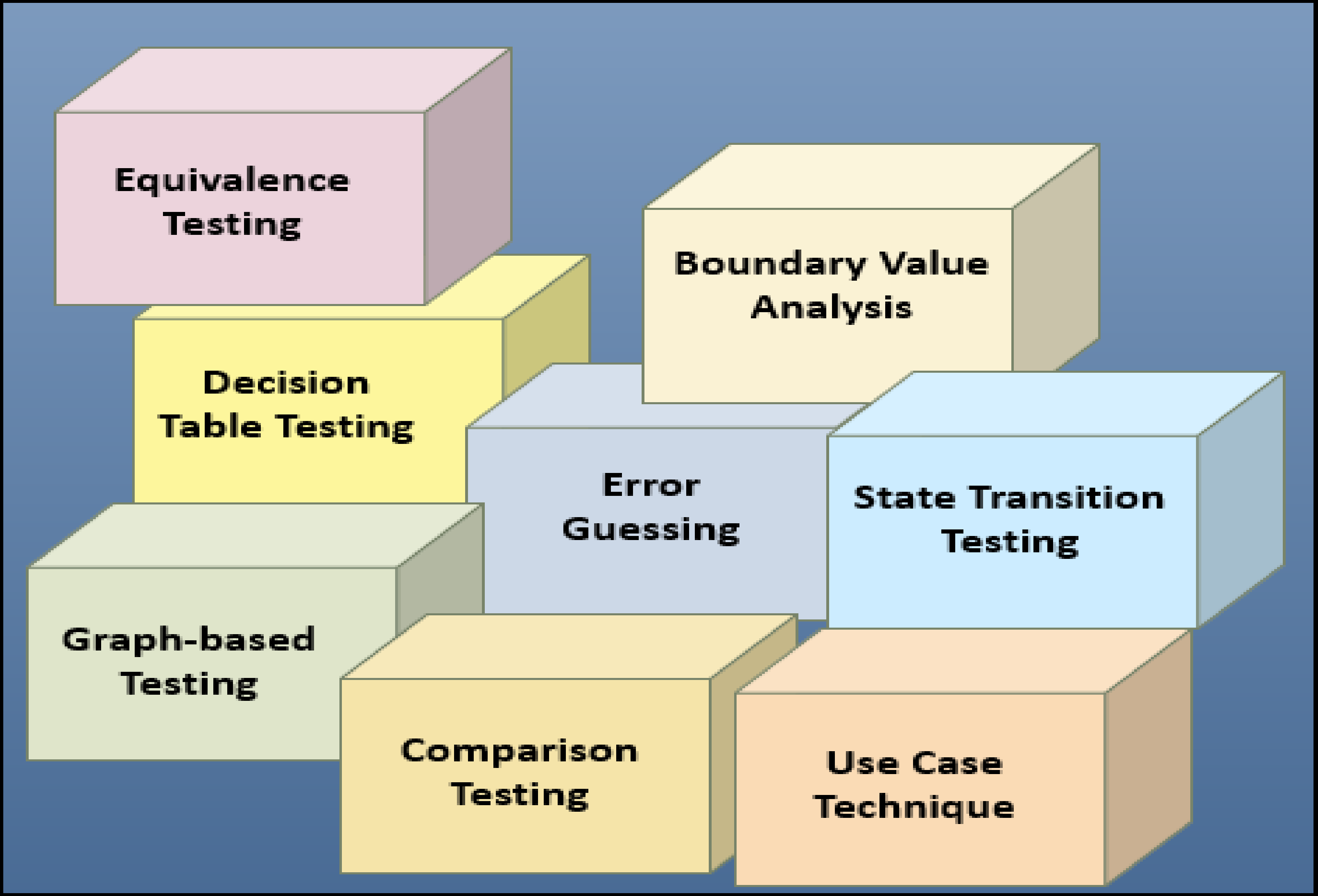
Dynamic Testing: Black-Box Testing

Black-box testing attempts to find errors in the following categories:

- **To test the functional validity of the software, so that incorrect or missing functions can be recognized.**

- **To test the maximum load or stress on the system.**

Black-Box Testing
[Test Case Design Techniques]



BOUNDARY VALUE ANALYSIS (BVA)

BOUNDARY VALUE ANALYSIS (BVA)

It is used for developing test cases to test the boundaries which separates the continuous range of inputs.

The boundaries are identified as part of this technique and then the test cases are written.

For any scenario, we can write:

- Lower boundary cases (using values just below the boundary specified)
- Upper Boundary cases (Using values just above the boundary specified)
- On boundary cases (Using the boundary values)

TYPES OF BOUNDARY VALUE ANALYSIS

BVA offers several methods to design test cases as listed below:

(1) BOUNDARY VALUE CHECKING [BVC]

(2) ROBUSTNESS TESTING METHOD [RTM]

(3) WORST-CASE TESTING METHOD [WTM]

(1) BOUNDARY VALUE CHECKING (BVC)

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:

(a) Minimum value (Min)

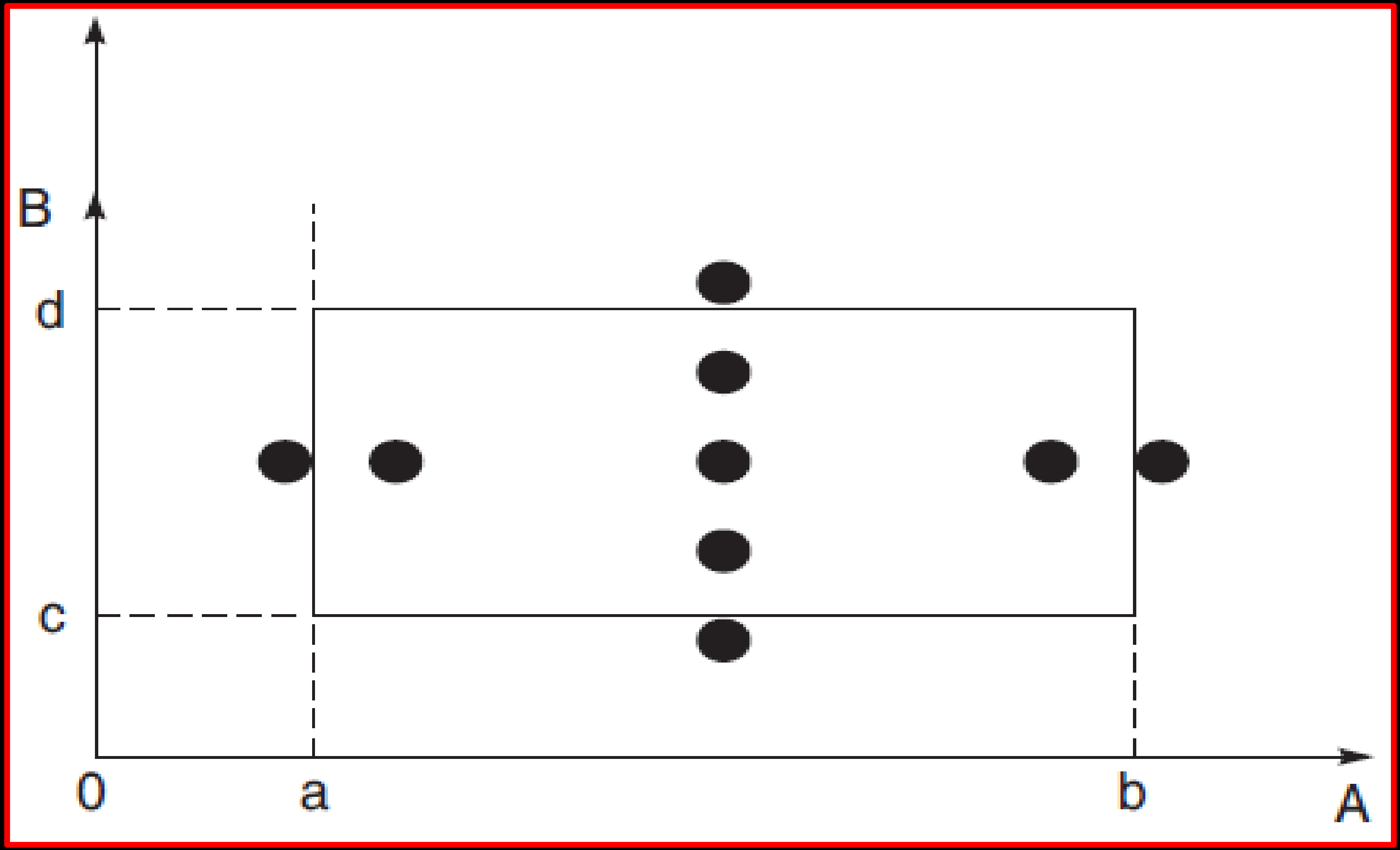
(b) Value just above the minimum value (Min+)

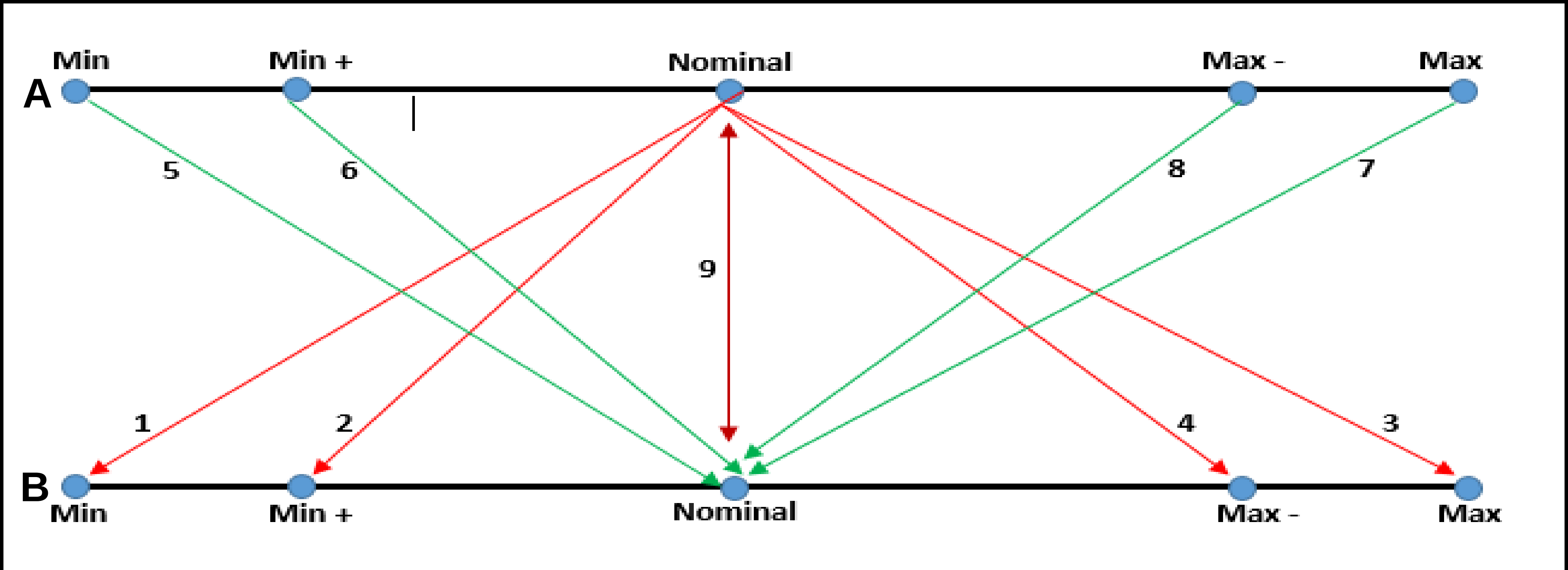
(c) Maximum value (Max)

(d) Value just below the maximum value (Max−)

Let us take the example of two variables, A and B.

If we consider all the below combinations with nominal values, then following test cases can be designed:





1. A_{nom}, B_{min}
2. A_{nom}, B_{min+}
3. A_{nom}, B_{max}
4. A_{nom}, B_{max-}
5. A_{min}, B_{nom}
6. A_{min+}, B_{nom}
7. A_{max}, B_{nom}
8. A_{max-}, B_{nom}
9. A_{nom}, B_{nom}

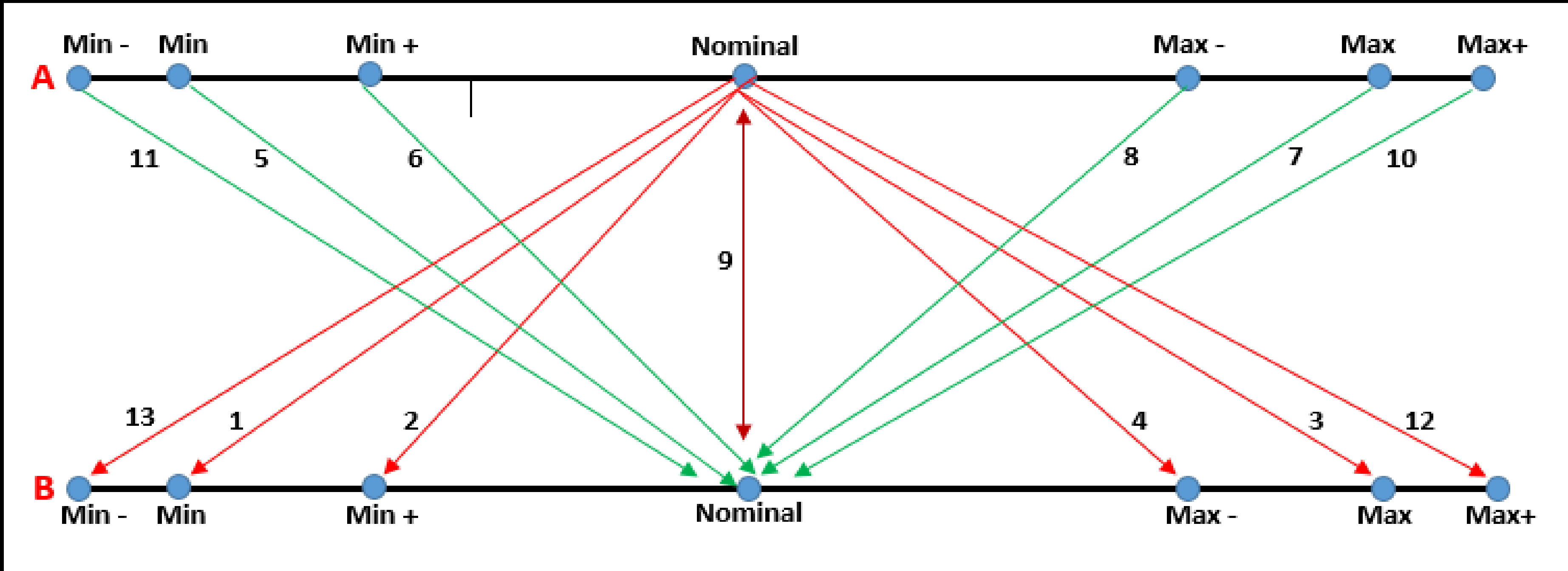
It can be generalized that for n variables in a module, 4n + 1 test cases can be designed with boundary value checking method.

(2) ROBUSTNESS TESTING METHOD

Test cases are designed considering the below points in addition to BVC, it is called robustness testing.

(a) A value just greater than the Maximum value (Max^+)

(b) A value just less than Minimum value (Min^-)



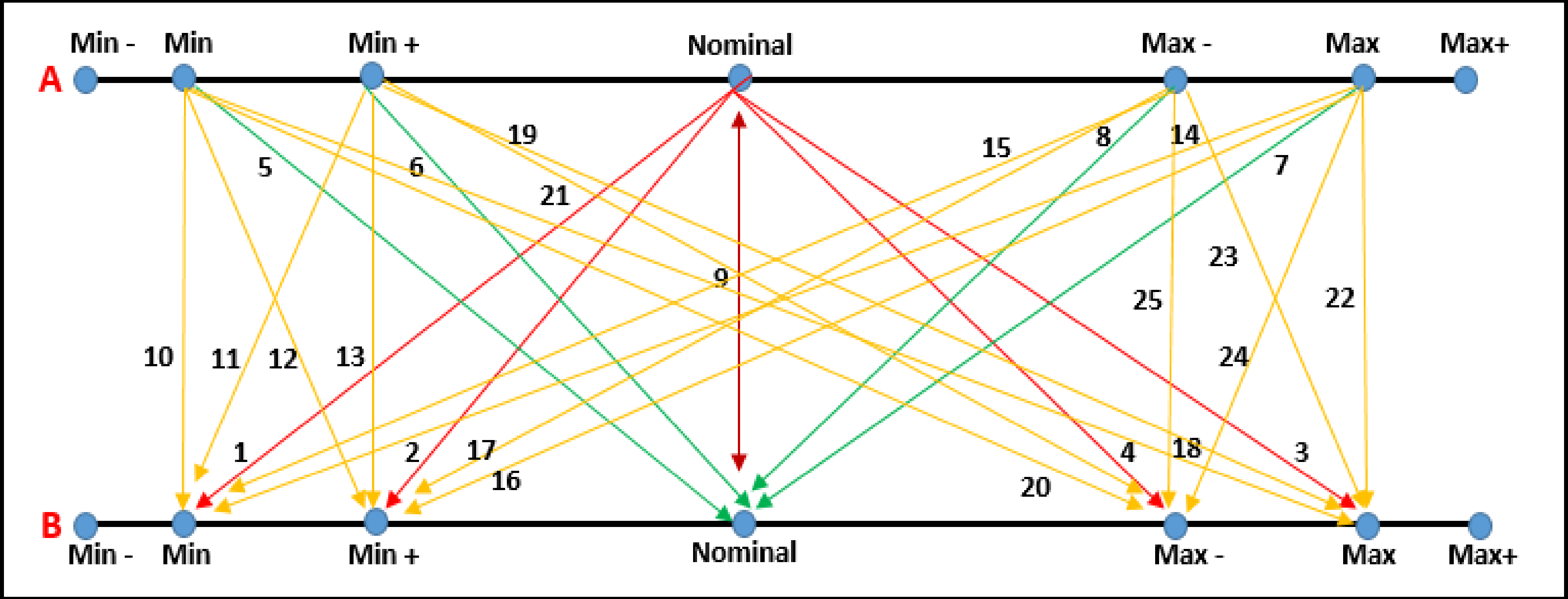
1. A_{nom}, B_{min}
2. A_{nom}, B_{min+}
3. A_{nom}, B_{max}
4. A_{nom}, B_{max-}
5. A_{min}, B_{nom}
6. A_{min+}, B_{nom}
7. A_{max}, B_{nom}
8. A_{max-}, B_{nom}
9. A_{nom}, B_{nom}
10. A_{max+}, B_{nom}
11. A_{min-}, B_{nom}
12. A_{nom}, B_{max+}
13. A_{nom}, B_{min-}

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

(3) WORST-CASE TESTING METHOD

We can again extend the concept of BVC by assuming more than one variable on the boundary.

We can add the following test cases to the list of 9 test cases designed in BVC as:



1. A_{nom}, B_{min}
2. A_{nom}, B_{min+}
3. A_{nom}, B_{max}
4. A_{nom}, B_{max-}
5. A_{min}, B_{nom}
6. A_{min+}, B_{nom}
7. A_{max}, B_{nom}
8. A_{max-}, B_{nom}
9. A_{nom}, B_{nom}
10. A_{min}, B_{min}
11. A_{min+}, B_{min}
12. A_{min}, B_{min+}
13. A_{min+}, B_{min+}
14. A_{max}, B_{min}
15. A_{max-}, B_{min}
16. A_{max}, B_{min+}
17. A_{max-}, B_{min+}
18. A_{min}, B_{max}
19. A_{min+}, B_{max}
20. A_{min}, B_{max-}
21. A_{min+}, B_{max-}
22. A_{max}, B_{max}
23. A_{max-}, B_{max}
24. A_{max}, B_{max-}
25. A_{max-}, B_{max-}

It can be generalized that for n variables in a module, 5^n test cases can be designed with robustness testing method.

Example -1

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.

(a) 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

(b) 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

(C) 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- 2

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.

(a) 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

(b) 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
12	5	6	Invalid input
13	5	3	125

(C) 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

Using these values, 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

Example- 3

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

$1 \leq mm \leq 12$
$1 \leq dd \leq 31$
$1900 \leq yyyy \leq 2025$

Its output would be the next date or it will display 'invalid date.'

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

(a) Test cases using BVC:

Since there are three variables, (month, day, and year) the total number of test cases will be 4n + 1 = 13. The set of boundary values is shown below:

	Month	Day	Year
Min value	1	1	1900
Min$^+$ value	2	2	1901
Max value	12	31	2025
Max$^-$ value	11	30	2024
Nominal value	6	15	1962

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

Test Case ID	Month	Day	Year	Expected Output
1	1	15	1962	16-1-1962
2	2	15	1962	16-2-1962
3	11	15	1962	16-11-1962
4	12	15	1962	16-12-1962
5	6	1	1962	2-6-1962
6	6	2	1962	3-6-1962
7	6	30	1962	1-7-1962
8	6	31	1962	Invalid input
9	6	15	1900	16-6-1900
10	6	15	1901	16-6-1901
11	6	15	2024	16-6-2024
12	6	15	2025	16-6-2025
13	6	15	1962	16-6-1962

(b) Test cases using robust testing:

The total number of test cases will be 6n + 1 = 19. The set of boundary values is shown below:

The set of boundary values is shown below:

	Month	Day	Year
Min^- value	0	0	1899
Min value	1	1	1900
Min^+ value	2	2	1901
Max value	12	31	2025
Max^- value	11	30	2024
Max^+ value	13	32	2026
Nominal value	6	15	1962

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

Test Case ID	Month	Day	Year	Expected Output
1	0	15	1962	Invalid date
2	1	15	1962	16-1-1962
3	2	15	1962	16-2-1962
4	11	15	1962	16-11-1962
5	12	15	1962	16-12-1962
6	13	15	1962	Invalid date
7	6	0	1962	Invalid date
8	6	1	1962	2-6-1962
9	6	2	1962	3-6-1962
10	6	30	1962	1-7-1962
11	6	31	1962	Invalid input
12	6	32	1962	Invalid date
13	6	15	1899	Invalid date
14	6	15	1900	16-6-1900
15	6	15	1901	16-6-1901
16	6	15	2024	16-6-2024
17	6	15	2025	16-6-2025
18	6	15	2026	Invalid date
19	6	15	1962	16-6-1962

(C) Test cases using **worst-case testing**:

The total number of test caseswill be $5^n = 125$.

The set of boundary values is shown below:

	Month	Day	Year
Min value	1	1	1900
Min^+ value	2	2	1901
Max value	12	31	2025
Max^- value	11	30	2024
Nominal value	6	15	1962

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

Test Case ID	Month	Day	Year	Expected Output
1	1	1	1900	2-1-1900
2	1	1	1901	2-1-1901
3	1	1	1962	2-1-1962
4	1	1	2024	2-1-2024
5	1	1	2025	2-1-2025
6	1	2	1900	3-1-1900
7	1	2	1901	3-1-1901
8	1	2	1962	3-1-1962
9	1	2	2024	3-1-2024
10	1	2	2025	3-1-2025
11	1	15	1900	16-1-1900
12	1	15	1901	16-1-1901
13	1	15	1962	16-1-1962
14	1	15	2024	16-1-2024
15	1	15	2025	16-1-2025
16	1	30	1900	31-1-1900

17	1	30	1901	31-1-1901
18	1	30	1962	31-1-1962
19	1	30	2024	31-1-2024
20	1	30	2025	31-1-2025
21	1	31	1900	1-2-1900
22	1	31	1901	1-2-1901
23	1	31	1962	1-2-1962
24	1	31	2024	1-2-2024
25	1	31	2025	1-2-2025
26	2	1	1900	2-2-1900
27	2	1	1901	2-2-1901
28	2	1	1962	2-2-1962
29	2	1	2024	2-1-2024
30	2	1	2025	2-2-2025
31	2	2	1900	3-2-1900
32	2	2	1901	3-2-1901
33	2	2	1962	3-2-1962
34	2	2	2024	3-2-2024
35	2	2	2025	3-2-2025
36	2	15	1900	16-2-1900
37	2	15	1901	16-2-1901
38	2	15	1962	16-2-1962
39	2	15	2024	16-2-2024
40	2	15	2025	16-2-2025
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
49	2	31	2024	Invalid date
50	2	31	2025	Invalid date
51	6	1	1900	2-6-1900
52	6	1	1901	2-6-1901
53	6	1	1962	2-6-1962
54	6	1	2024	2-6-2024
55	6	1	2025	2-6-2025
56	6	2	1900	3-6-1900
57	6	2	1901	3-6-1901
58	6	2	1962	3-6-1962
59	6	2	2024	3-6-2024
60	6	2	2025	3-6-2025
61	6	15	1900	16-6-1900
62	6	15	1901	16-6-1901
63	6	15	1962	16-6-1962
64	6	15	2024	16-6-2024
65	6	15	2025	16-6-2025
66	6	30	1900	1-7-1900
67	6	30	1901	1-7-1901
68	6	30	1962	1-7-1962
69	6	30	2024	1-7-2024
70	6	30	2025	1-7-2025
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	2-11-1900
77	11	1	1901	2-11-1901
78	11	1	1962	2-11-1962
79	11	1	2024	2-11-2024
80	11	1	2025	2-11-2025
81	11	2	1900	3-11-1900
82	11	2	1901	3-11-1901
83	11	2	1962	3-11-1962
84	11	2	2024	3-11-2024
85	11	2	2025	3-11-2025
86	11	15	1900	16-11-1900
87	11	15	1901	16-11-1901
88	11	15	1962	16-11-1962
89	11	15	2024	16-11-2024
90	11	15	2025	16-11-2025
91	11	30	1900	1-12-1900
92	11	30	1901	1-12-1901

93	11	30	1962	1-12-1962
94	11	30	2024	1-12-2024
95	11	30	2025	1-12-2025
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
101	12	1	1900	2-12-1900
102	12	1	1901	2-12-1901
103	12	1	1962	2-12-1962
104	12	1	2024	2-12-2024
105	12	1	2025	2-12-2025
106	12	2	1900	3-12-1900
107	12	2	1901	3-12-1901
108	12	2	1962	3-12-1962
109	12	2	2024	3-12-2024
110	12	2	2025	3-12-2025
111	12	15	1900	16-12-1900
112	12	15	1901	16-12-1901
113	12	15	1962	16-12-1962
114	12	15	2024	16-12-2024
115	12	15	2025	16-12-2025
116	12	30	1900	31-12-1900
117	12	30	1901	31-12-1901
118	12	30	1962	31-12-1962
119	12	30	2024	31-12-2024
120	12	30	2025	31-12-2025
121	12	31	1900	1-1-1901
122	12	31	1901	1-1-1902
123	12	31	1962	1-1-1963
124	12	31	2024	1-1-2025
125	12	31	2025	1-1-2026

EQUIVALENCE CLASS TESTING

EQUIVALENCE CLASS PARTITIONING

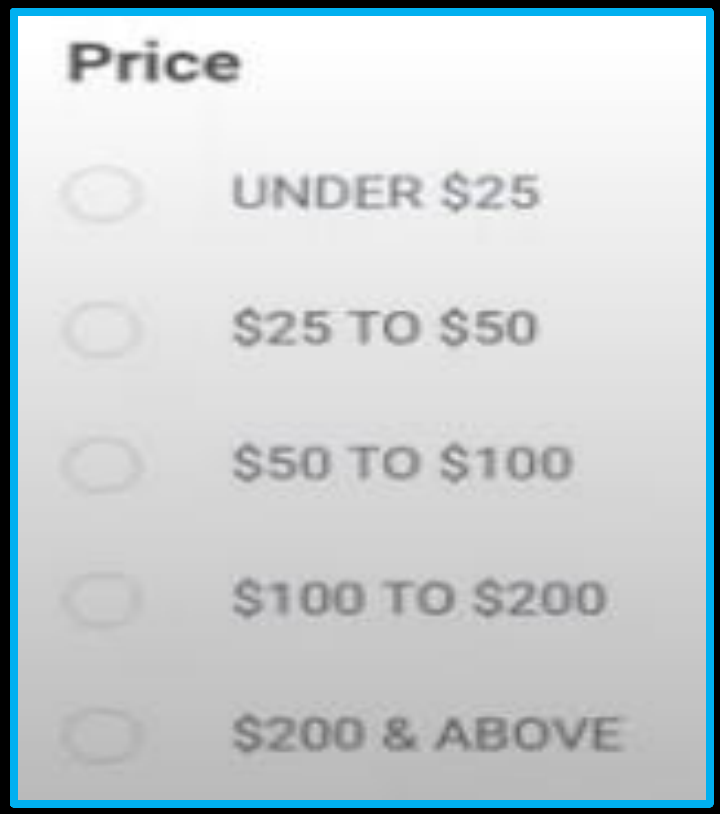
Equivalence partitioning is also known as "**Equivalence Class Partitioning**". The whole range of input data for a given requirement are grouped into several sets or equivalence classes.

Equivalence classes are determined in such a way that all the input values in a input data range produce the same output when it is processed by the software.

EQUIVALENCE CLASS PARTITIONING

Examples :

Price filter for any ecommerce site is already designed with Equivalence partitioning in mind.



Arivuselvan.K

EQUIVALENCE CLASS PARTITIONING

University Grading systems.

Grade	Grade Points	Mark Range
O	10	91-100
A+	9	81-90
A	8	71-80
B+	7	61-70
B	6	50-60
RA	0	0 - 49 or <50

One more additional test case(For invalid data) is to test values which is greater than 100 and it should not be permitted.

Equivalence partitioning method for designing test cases has the following goals:

(1) Completeness:

Without executing all the test cases, we strive to touch the completeness of testing domain.

(2) Non-redundancy:

When the test cases are executed having inputs from the same class, then there is redundancy in executing the test cases.

Time and resources are wasted in executing these redundant test cases, as they explore the same type of bug.

To use equivalence partitioning, one needs to perform two steps:

(1) Identify equivalence classes

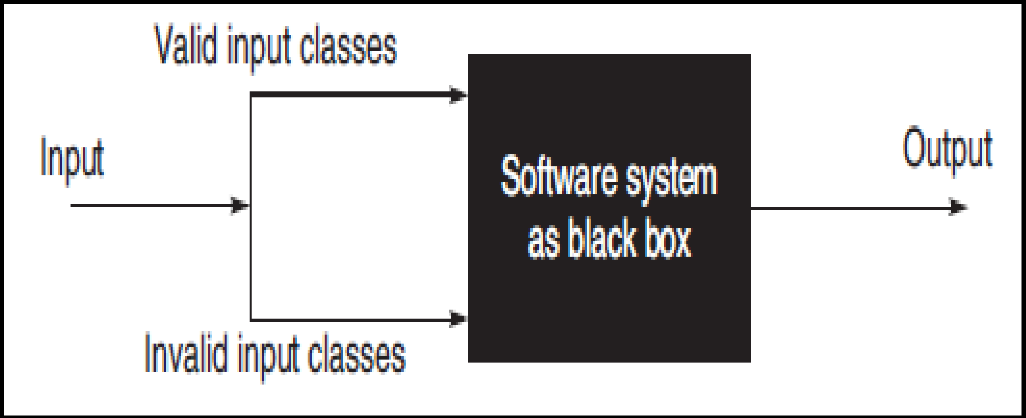
(2) Design test cases

(1) Identification Of Equivalent Classes:

Two types of classes can always be identified:

(i) Valid equivalence classes: These classes consider valid inputs to the program.

(ii) Invalid equivalence classes: Consider invalid inputs that will generate error conditions or unexpected behavior of the program



There are no well-defined rules for identifying equivalence classes, as it is a heuristic process.

However, some guidelines are defined for forming equivalence classes:

(i) Assign a unique identification number to each equivalence class.

(ii) Write a new test case. [covering as many of the uncovered valid equivalence classes as possible]

EQUIVALENCE PARTITION

Example:1

DATE (1 To 31)

Invalid Partition	Valid Partition	Invalid Partition
0 -1 -2 ….	1 2 3 …. 31	32 33 34 …

Example:2

USER NAME (6 To 10 Characters)

Invalid Partition	Valid Partition	Invalid Partition
0 Character 1 Character 2 Character 3 Character 4 Character 5 Character	6 Character 7 Character 8 Character 9 Character 10 Character	11 Character 12 Character …

Example:3

AGE (18 TO 80 Years, EXCEPT 60 To 65 Years)

Invalid Partition	Valid Partition	Invalid Partition	Valid Partition	Invalid Partition
0 year 1 year …… 17 years	18 years …. 59 years	60 years …… 65 years	66 years ……. 80 years	81 years 82 years ……

EQUIVALENCE CLASS TESTING – Example 1

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:

$$I_1 = \{<A,B,C> : 1 \le A \le 50\}$$
$$I_2 = \{<A,B,C> : 1 \le B \le 50\}$$
$$I_3 = \{<A,B,C> : 1 \le C \le 50\}$$
$$I_4 = \{<A,B,C> : A < 1\}$$
$$I_5 = \{<A,B,C> : A > 50\}$$
$$I_6 = \{<A,B,C> : B < 1\}$$
$$I_7 = \{<A,B,C> : B > 50\}$$
$$I_8 = \{<A,B,C> : C < 1\}$$
$$I_9 = \{<A,B,C> : C > 50\}$$

Now the **test cases** can be designed from the above **derived classes**, taking **one test case** from **each class**.

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

$$I_1 = \{\langle A,B,C\rangle : 1 \le A \le 50\}$$
$$I_2 = \{\langle A,B,C\rangle : 1 \le B \le 50\}$$
$$I_3 = \{\langle A,B,C\rangle : 1 \le C \le 50\}$$
$$I_4 = \{\langle A,B,C\rangle : A < 1\}$$
$$I_5 = \{\langle A,B,C\rangle : A > 50\}$$
$$I_6 = \{\langle A,B,C\rangle : B < 1\}$$
$$I_7 = \{\langle A,B,C\rangle : B > 50\}$$
$$I_8 = \{\langle A,B,C\rangle : C < 1\}$$
$$I_9 = \{\langle A,B,C\rangle : C > 50\}$$

2. We can **derive another set of equivalence classes** based on some possibilities for three integers, **A, B, and C**. These are given below:

$$I_1 = \{<A,B,C> : A > B, A > C\}$$
$$I_2 = \{<A,B,C> : B > A, B > C\}$$
$$I_3 = \{<A,B,C> : C > A, C > B\}$$
$$I_4 = \{<A,B,C> : A = B, A \neq C\}$$
$$I_5 = \{<A,B,C> : B = C, A \neq B\}$$
$$I_6 = \{<A,B,C> : A = C, C \neq B\}$$
$$I_7 = \{<A,B,C> : A = B = C\}$$

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

EQUIVALENCE CLASS TESTING
Example-2

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

$$1 \le mm \le 12$$

$$1 \le dd \le 31$$

$$1900 \le yyyy \le 2025$$

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:

$$I_1 = \{<m, d, y> : 1 \le m \le 12\}$$
$$I_2 = \{<m, d, y> : 1 \le d \le 31\}$$
$$I_3 = \{<m, d, y> : 1900 \le y \le 2025\}$$
$$I_4 = \{<m, d, y> : m < 1\}$$
$$I_5 = \{<m, d, y> : m > 12\}$$
$$I_6 = \{<m, d, y> : d < 1\}$$
$$I_7 = \{<m, d, y> : d > 31\}$$
$$I_8 = \{<m, d, y> : y < 1900\}$$
$$I_9 = \{<m, d, y> : y > 2025\}$$

The test cases can be designed from the above derived classes are shown below:

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

Equivalence class VS Boundary Value Analysis

Equivalence Class Partitioning	Boundary Value Analysis
Valid & Invalid equivalence classes (ranges) of the input data domain are considered for test design & execution.	It considers the following for input data values: • Minimum – 1 • Minimum • Minimum + 1 • Nominal • Maximum - 1 • Maximum • Maximum + 1
It considers input data values from a range of equivalence classes (intervals) of the input data domain.	It considers input data values based on the defined boundaries of the input data domain.
It is significant in identifying bugs in-between the defined equivalence classes.	It is significant in identifying bugs at the boundaries of the input data domain.