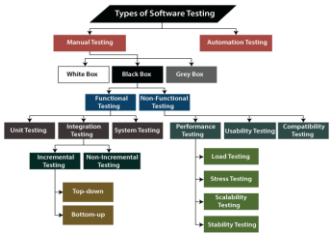
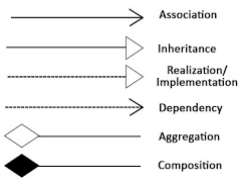
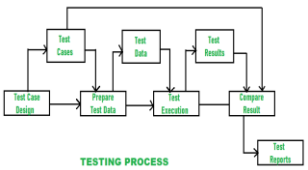
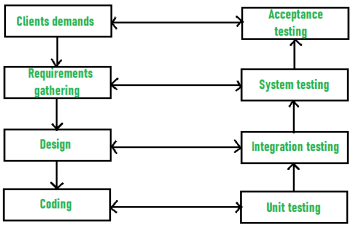



# Software Design and Testing

# *Dynamic Testing: Black box Testing*




*Chapter 4*

## Outline

- Boundary Value Analysis
- Equivalence Class Testing

## Introduction

- The term 'Black Box' refers to the software, which is treated as a black box.
- By treating it as a black box, we mean that the system or source code is not checked at all.
- It is done from customer's viewpoint.
- The test engineer engaged in black box testing only knows the set of inputs and expected outputs and is unaware of how those inputs are transformed into outputs by the software.

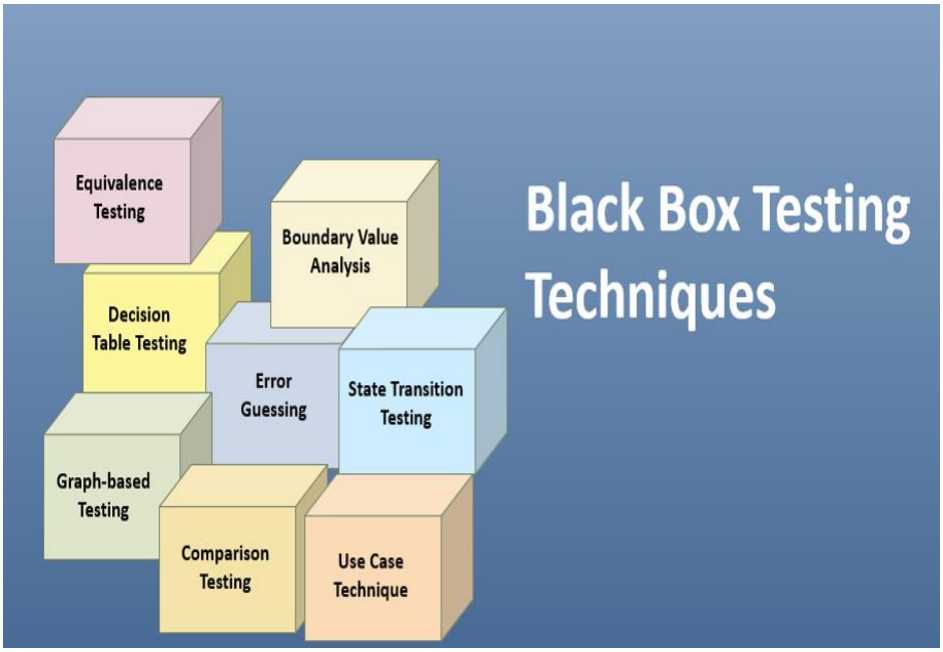

*Internal behavior of the code is unknown*

3

## Black box testing

- Black-box testing attempts to find errors in the following categories:
  - To test the modules independently.
  - To test the functional validity of the software
  - Interface errors are detected.
  - To test the system behavior and check its performance.
  - To test the maximum load or stress on the system.
  - Customer accepts the system within defined acceptable limits.

4

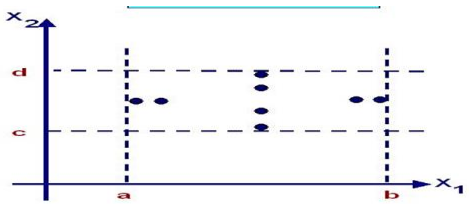



## Boundary Value Analysis (BVA)

- This Black Box testing technique believes and extends the concept that the density of defect is more towards the boundaries. This is done due to the following reasons
  - Usually the programmers are not able to decide whether they have to use <= operator or < operator when trying to make comparisons.
  - Different terminating conditions of For-loops, While loops and Repeat loops may cause defects to move around the boundary conditions.
  - The requirements themselves may not be clearly understood, especially around the boundaries, thus causing even the correctly coded program to not perform the correct way.

## Boundary Value Analysis (BVA)

- The basic idea of BVA is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum and at their maximum. Meaning thereby (min, min+, nom, max-, max), as shown in the following figure

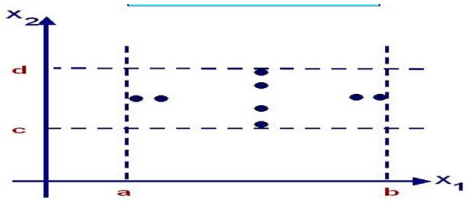



- Example: A textbox can input integer numbers from 1 to 99
- Select value for test case: 1; 2; 98; 99


## Boundary Value Analysis (BVA)

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

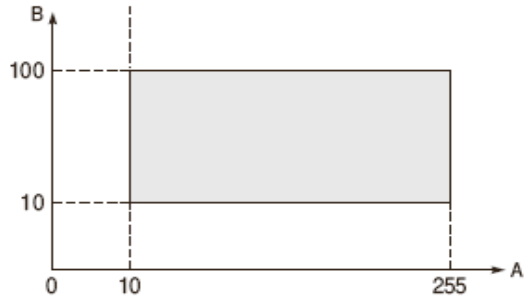
1. Minimum value (min)
2. Value just above the minimum value ($min^+$)
3. Maximum Value (max)
4. Value just below the maximum value ($max^-$)
5. Nominal Value (nom)




8

## Boundary Value Analysis (BVA)

- Example : if A is an integer between 10 and 255, then boundary checking can be on 10(9,10,11) and on 255(256,255,254).
- Similarly, B is another integer variable between 10 and 100, then boundary checking can be on 10(9,10,11) and 100(99,100,101), as shown in Fig.




9

## Boundary Value Analysis (BVA)

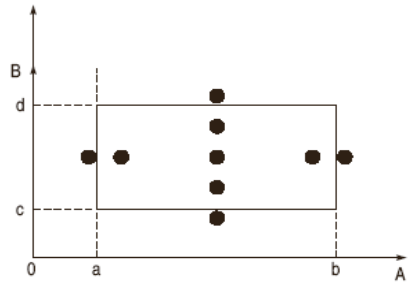
- BVA is based upon a critical assumption that is known as "**Single fault assumption theory**". According to this assumption, we derive the test cases on the basis of the fact that failures are not due to simultaneous occurrence of two (or more) faults. So, we derive test cases by holding the values of all but one variable at their nominal values and allowing that variable assume its extreme values.
- If we have a function of **n-variables**, we hold all but one at the nominal values and let the remaining variable assume the **min, min+, nom, max-and max values**, repeating this for each variable.
- Thus, for a function of **n** variables, BVA yields **(4n + 1)** test cases.

10

## Boundary Value Analysis (BVA)

- Consider 2 inputs to a program

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. $A_{nom}$, $B_{nom}$




- 4n+1 test cases can be designed with boundary value checking method.

11

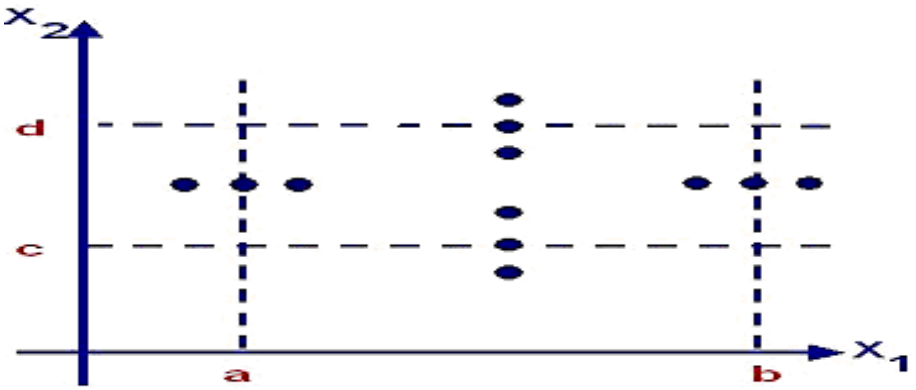
## Robustness Testing

- Robustness Testing is another variant of BVA
- In BVA, we remain within the legitimate boundary of our range i.e. for testing we consider values like **(min, min+, nom, max-, max)** whereas in Robustness testing, we try to cross these legitimate boundaries as well.
- Thus for testing here we consider the values like **(min-, min, min+, nom, max-, max, max+)**
- Again, with robustness testing, we can focus on exception handling. With strongly typed languages, robustness testing may be very awkward. For example, in PASCAL, if a variable is defined to be within a certain range, values outside that range result in run-time errors thereby aborting the normal execution.

12

## Robustness Testing

- For a program with **n**-variables, robustness testing will yield **(6n + 1)** test-cases. Thus we can draw the following Robustness Test Cases graph




13

## Robustness Testing

- A value just greater than the Maximum value ($Max^+$)
- A value just less than Minimum value ($Min^-$)
- When test cases are designed considering above points in addition to BVC, it is called Robustness testing.
- For n=2 we have 13 testcases.

10. $A_{max+}$, $B_{nom}$
11. $A_{min-}$, $B_{nom}$
12. $A_{nom}$, $B_{max+}$
13. $A_{nom}$, $B_{min-}$

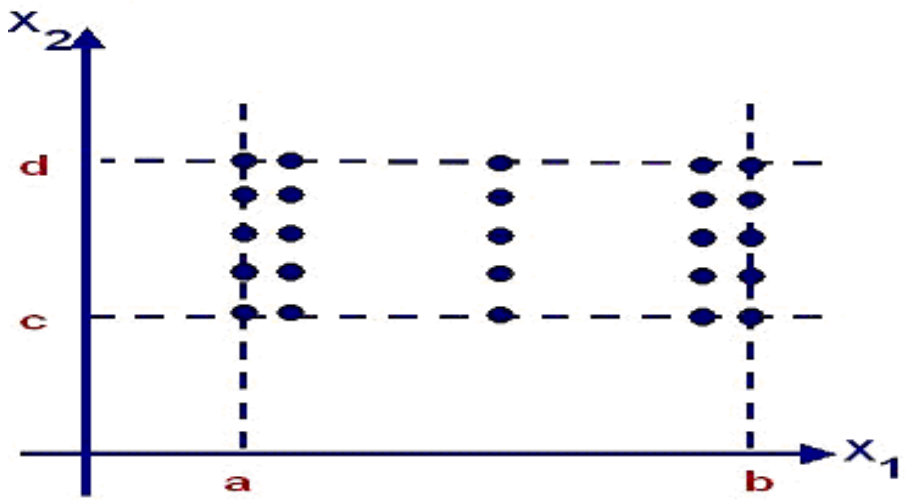
14

## Worst Case Testing

- If we reject our basic assumption of single fault assumption theory and focus on what happens when we reject this theory-it simply means that we want to see that what happens when more than one variable has an extreme value. In electronic circuit analysis, this is called as "worst-case analysis". We use this idea here to generate worst-case test cases.
- For each variable, we start with the five-element set that contains the **min, min+, nom, max-, and max** values.
- We then take the Cartesian product of these sets to generate test cases. This is shown in following graph.
- For a program with **n**-variables, $\mathbf{5^n}$ test cases are generated

15

**Worst Case Test Case**




8-16

## Worst Case Testing

- When more than one variable are in extreme values, i.e. when more than one variable are on the boundary. It is called Worst case testing method.
- It can be generalized that for n input variables in a module, $5^n$ test cases are designed with worst case testing.
- For n = 2 we have 25 test cases

| | |
|---|---|
| 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. $A_{max-}$, $B_{max-}$ |

17

## Worst Case Testing

- When more than one variable are in extreme values, i.e. when more than one variable are on the boundary. It is called Worst case testing method.
- It can be generalized that for n input variables in a module, $5^n$ test cases are designed with worst case testing.
- For n = 2 we have 25 test cases

| | |
|---|---|
| 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. $A_{max-}$, $B_{max-}$ |

18

## Guidelines For BVA

1. The normal versus robust values and the single fault versus the multiple-fault assumption theory result in better testing. These methods can be applied to both input and output domain of any program.
2. Robustness testing is a good choice for testing internal variables.
3. We must bear in mind that we can create extreme boundary results from non-extreme input values

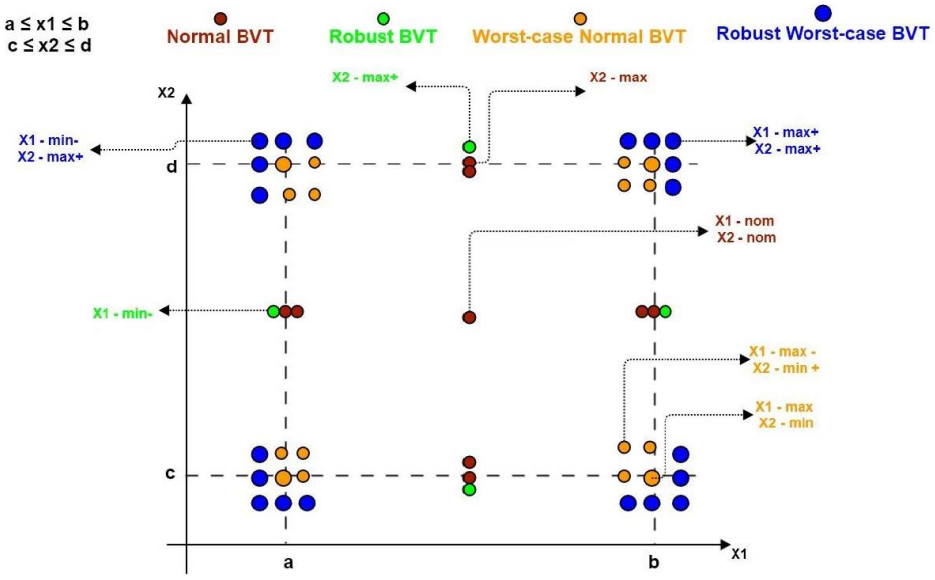
1. **Normal Boundary Value Testing: 4n+1**
2. **Robust Boundary Value Testing: 6n+1**
3. **Worst-Case Boundary Value Testing: 5^n**
4. **Robust worst-case boundary value testing: 7^n**

19

## Limitations of Boundary Value Analysis (BVA)

- Boolean and logical variables present a problem for Boundary Value Analysis
- BVA assumes the variables to be truly independent which is not always possible.
- BVA test cases have been found to be rudimentary because they are obtained with very little insight and imagination.

20




## Equivalence Partitioning

- Inputs to the software are divided into groups that are expected to exhibit similar behavior, so they are likely to be processed in the same way.
- Equivalence partitions can be found for
  - Valid data – values that should be accepted.
  - Invalid data – values that should be rejected.
- Example: A textbox can input integer numbers from 1 to 99
  - Partition 1 – invalid: < 1
  - Partition 2 – valid: 1 <= & <=99
  - Partition 3 – invalid: > 99


- Equivalence Partitioning is also known as Equivalence Class Partitioning.

22

## Equivalence Partitioning

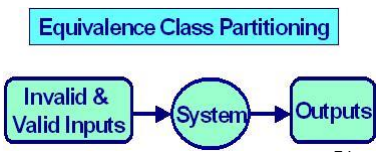
- The idea of equivalence class testing is to identify test cases by using one element from each equivalence class.
- If the equivalence classes are chosen wisely, the potential redundancy among test cases can be reduced.
- If one test case in an equivalence class detects a bug, all other test cases in that class have the same probability of finding bugs.
- Therefore, instead of taking every value in one domain, only one test case is chosen from one class.

23

## Equivalence Partitioning Goals

- **Completeness** : Without executing all the test cases, we strive to touch the completeness of testing domain.
- **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.
- The goal of equivalence partitioning method is to reduce these redundant test cases.
- To use equivalence partitioning, one needs to perform two steps:

1. Identify equivalence classes
2. Design test cases



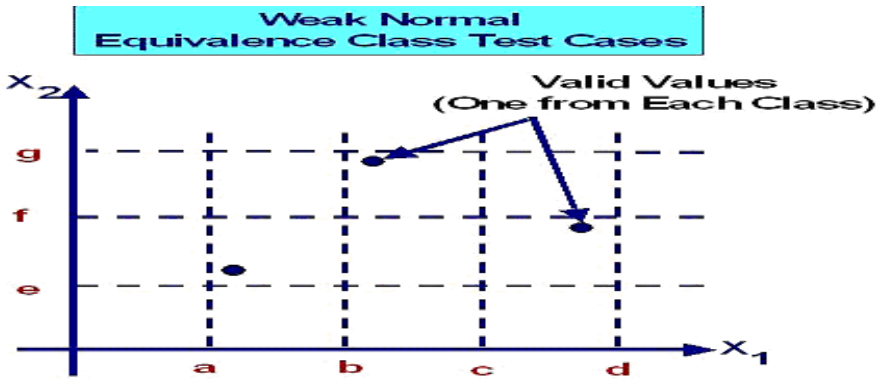
## Types of Equivalence Partitioning

1. Weak Normal Equivalence Class Testing.
2. Strong Normal Equivalence Class Testing.
3. Weak Robust Equivalence Class Testing.
4. Strong Robust Equivalence Class Testing.



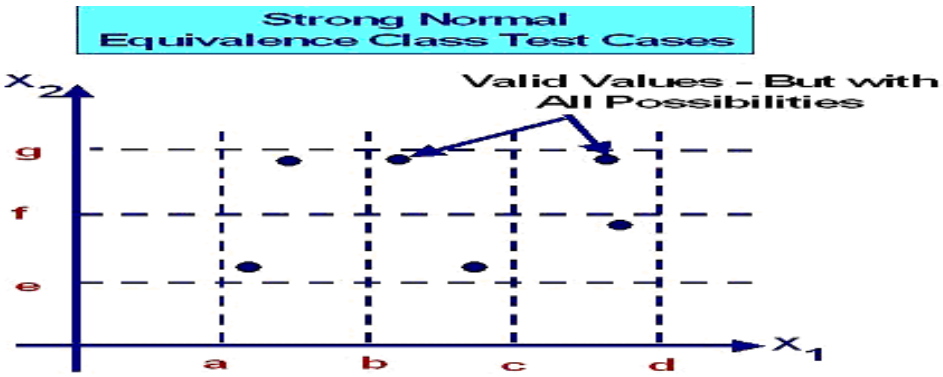
## Weak Normal Equivalence Class Testing

- The word 'weak' means 'single fault assumption'.
- This type of testing is accomplished by using one variable from each equivalence class in a test case. [valid values]
- We would, thus, end up with the weak equivalence class test cases as shown in the following figure



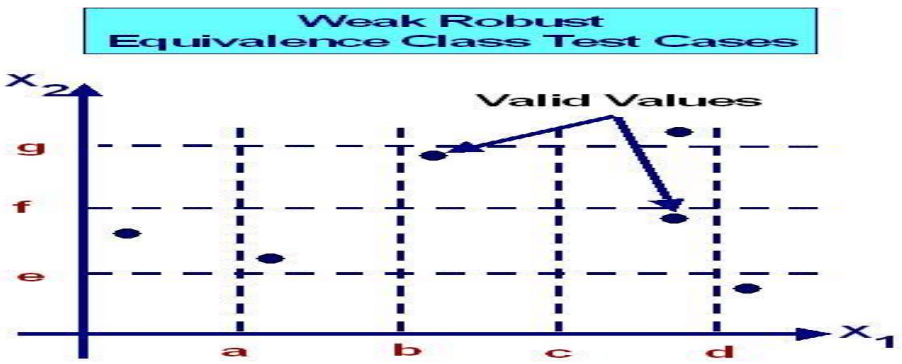
## Strong Normal Equivalence Class Testing

- This type of testing is based on the multiple fault assumption theory. So, now we need test cases from each element of the Cartesian product [like truth table] of the equivalence classes, as shown in the following figure.



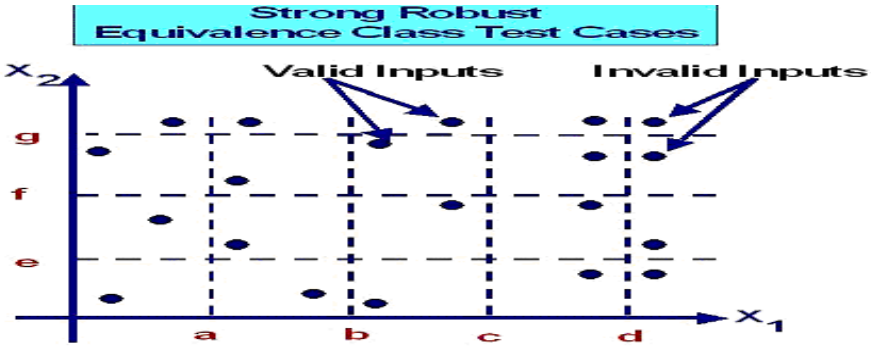

## Weak Robust Equivalence Class Testing

- The word' weak' means single fault assumption theory and the word 'Robust' refers to invalid values.
- The test cases resulting from this strategy are shown in the following figure



## Strong Robust Equivalence Class Testing

- As explained earlier also, 'robust' means consideration of invalid values and the 'strong' means multiple fault assumption.
- We obtain the test cases from each element of the Cartesian product of all the equivalence classes as shown in the following figure



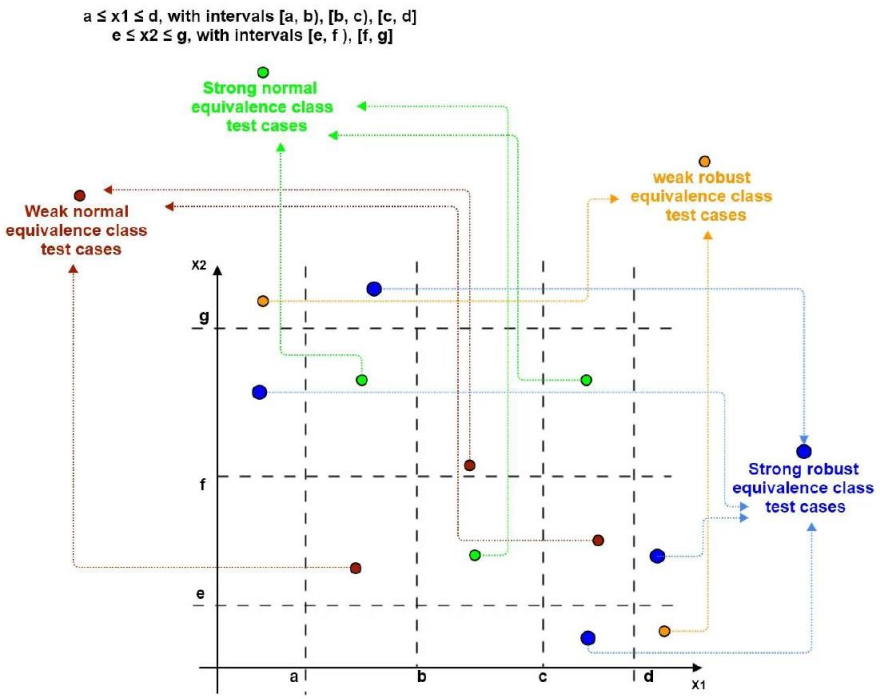

## Guidelines for Equivalence Class Testing

1. The weak forms of equivalence class testing (normal or robust) are not as comprehensive as the corresponding strong forms.
2. If the implementation language is strongly typed and invalid values cause run-time errors then there is no point in using the robust form.
3. If error conditions are a high priority, the robust forms are appropriate.
4. Equivalence class testing is approximate when input data is defined in terms of intervals and sets of discrete values. This is certainly the case when system malfunctions can occur for out-of-limit variable values.
5. Equivalence class testing is strengthened by a hybrid approach with boundary value testing (BVA).

## Guidelines for Equivalence Class Testing

6. Equivalence class testing is used when the program function is complex. In such cases, the complexity of the function can help identify useful equivalence classes.
7. Strong equivalence class testing makes a presumption that the variables are independent and the corresponding multiplication of test cases raises issues of redundancy. If any dependencies occur, they will often generate "error" test cases.
8. Several tries may be needed before the "right" equivalence relation is established.
9. The difference between the strong and weak forms of equivalence class testing is helpful in the distinction between progression and regression testing.

31



## Difference between ECT & BVA

| *Equivalence Class Testing* | *Boundary Value Analysis* |
|---|---|
| 1. Equivalence Class Testing is a type of black box technique. | 1. Next part of Equivalence Class Partitioning/Testing. |
| 2. It can be applied to any level of testing, like unit, integration, system, and more. | 2. Boundary value analysis is usually a part of **stress** & **negative testing**. |
| 3. A **test case design technique** used to divide input data into different equivalence classes. | 3. This test case design technique used to test boundary value between partitions. |
| 4. Reduces the time of testing, while using less and effective test cases. | 4. Reduces the overall time of test execution, while making **defect detection** faster & easy. |
| 5. Tests only one from each partition of the equivalence classes. | 5. Selects test cases from the edges of the equivalence classes. |

## Advantages and Disadvantages of Black Box

| Advantages | Disadvantages |
|---|---|
| Code knowledge is not required, tester's perception is very simple | Limited coverage, few test scenarios are designed/performed. |
| User's and developer's view are separate | Some parts of the backend are not tested at all. |
| Access to code is unrequired, quicker test case development | Inefficient testing due to the limited knowledge of code possesses by a tester. |
| Efficient and suitable for large parts of code | Test cases are difficult to design without clear specification |

## Some of tools for Black Box

| | | |
|---|---|---|
| Auto Hotkey | JMeter | Appium |
| OWASP ZEBEDIA | Auto Italia Online | Selenium IDE |
| Ranorex | Katalon | MbUnit |
| QTP/ UFT | SilkTest | Water |
| Selendroid | Gremlins | IBM Rational Functional tester |
| Squish by froglogic | Aegis Web | Applitools |
| Microsoft Coded UI | HP QTP | |

35


36