# SOFTWARE TESTING TECHNIQUES

## (A) BLACK BOX TESTING TECHNIQUES

# Learning Goals for Today

- **Black Box Testing**
- **Types of Black Box Testing**
- **Black Box Testing Techniques**
  - Boundary Value Analysis
  - Equivalence Partitioning
  - Decision Tree Testing
  - State Transition Testing
  - Error Guessing
  - Graph-Based Testing Methods
  - Comparison Testing
- **Develop a test suite using real live case study**
- **Evaluate the strengths and weaknesses of Black-box and White-box Testing**

# Black Box Testing

- **Recall:**
- **Most of us perform Black Box Testing every day!**
- Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as **Specifications based testing. I,e testing the functional requirements of the system.**
- **It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested.**
- Black box testing is perform from the beginning of the SDLC.
- Design test cases from requirement specification only (no knowledge of code structure)
  - Tester must understand the user perspective (domain knowledge)
  - Testing is done based on user's view point
- **The main advantage** of black box box testing is that, the tester do not need to have knowledge on a specific programming language or any implementation detail

## Various parameters checked in black box testing are:

- Accurate actions performed by users
- System’s interaction with the inputs
- The response time of the system
- Usability issues
- Performance issues
- Abrupt application failure, unable to start or finish

## There are many different types of Black Box Testing, some of them are given below

**(1) Functional testing – This is a type of black box testing which is related to the functional requirements of a system; Functional testing is concerned only with the functional requirements of a system and covers how well the system executes its functions.**

- **Functional testing is concerned only with the functional requirements of a system or subsystem and covers how well (if at all) the system executes its functions. These include any user commands, data manipulation, searches and business processes, user screens, and integration**
- **Functional testing is done using the functional specifications provided by the client or by using the design specifications like use cases provided by the design team**

# (2) Non functional testing

- Non functional testing is concerned with the non-functional requirements and is designed specifically to evaluate the readiness of a system according to the various criteria **(Quality Attributes)** which are not covered by functional testing
- e.g
- Performance Testing
- Security Testing
- Usability Testing
- Reliability and Dependability Testing
- Endurance testing

## (3) Regression testing

- Regression testing – Regression Testing is performed after code fixes, upgrades or any other system maintenance to check the new changes has not affected any existing functionality.

## Black Box Testing techniques

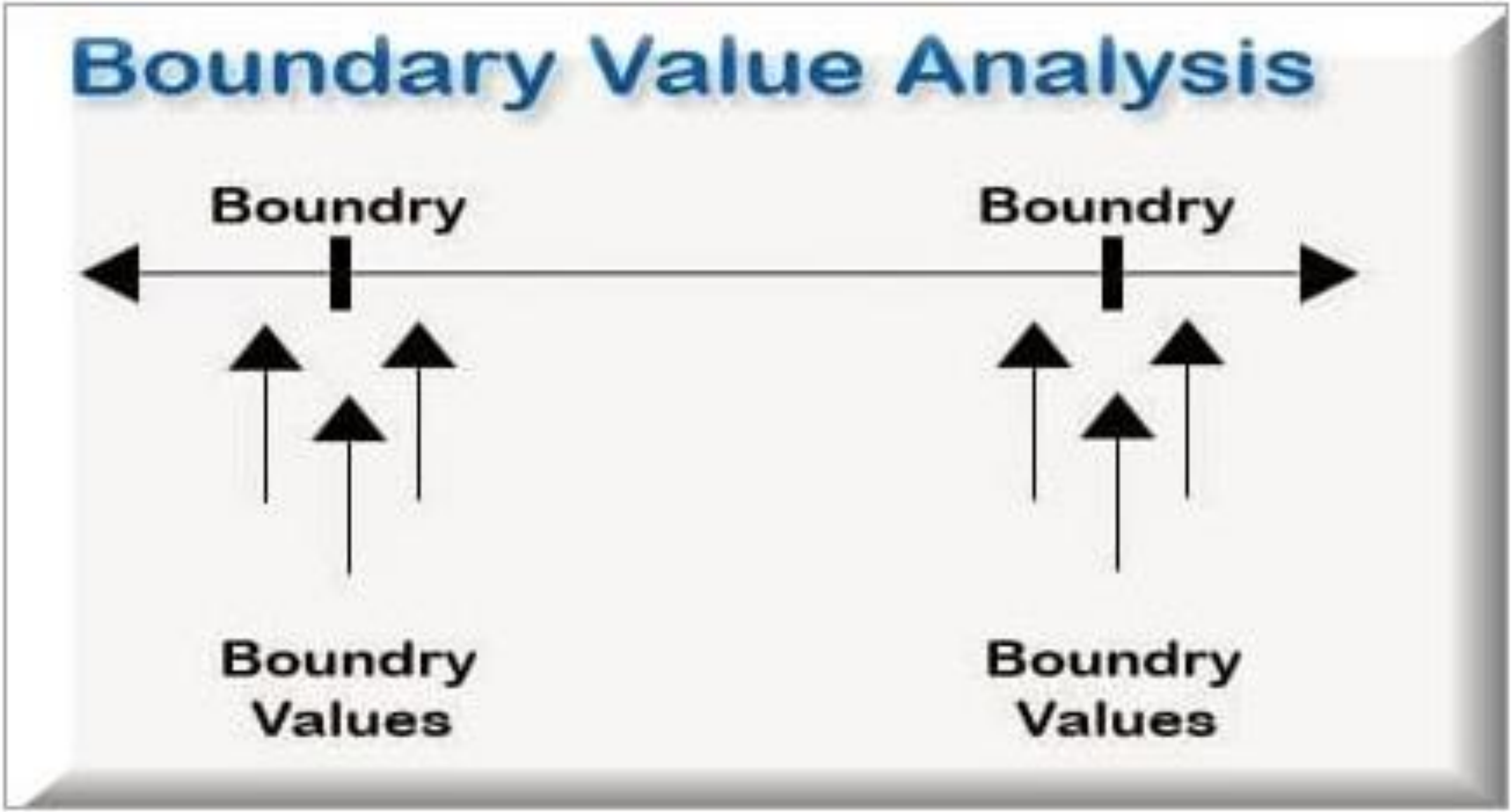
- In order to systematically test a set of functions, it is necessary to design test cases. Testers can create test cases from the requirement specification document using the following Black Box Testing techniques:

## 1) Boundary Value Analysis (BVA)

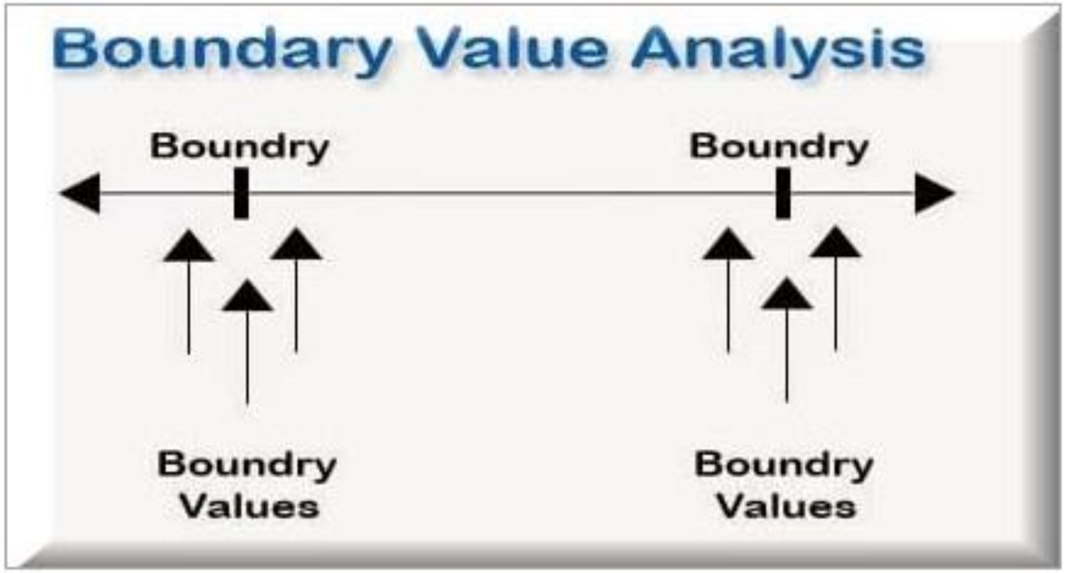
- It is the widely used black-box testing, which is also the **basis for equivalence testin**g. Boundary value analysis tests the software with test cases with extreme values of test data. BVA is used to identify the flaws or errors that arise due to the limits of input data

- The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.

- Boundary refers to **values near the limit** where the behavior of the system changes. In boundary


Boundary Value Analysis
Boundry
Boundry
Boundry Values
Boundry Values

## For Example



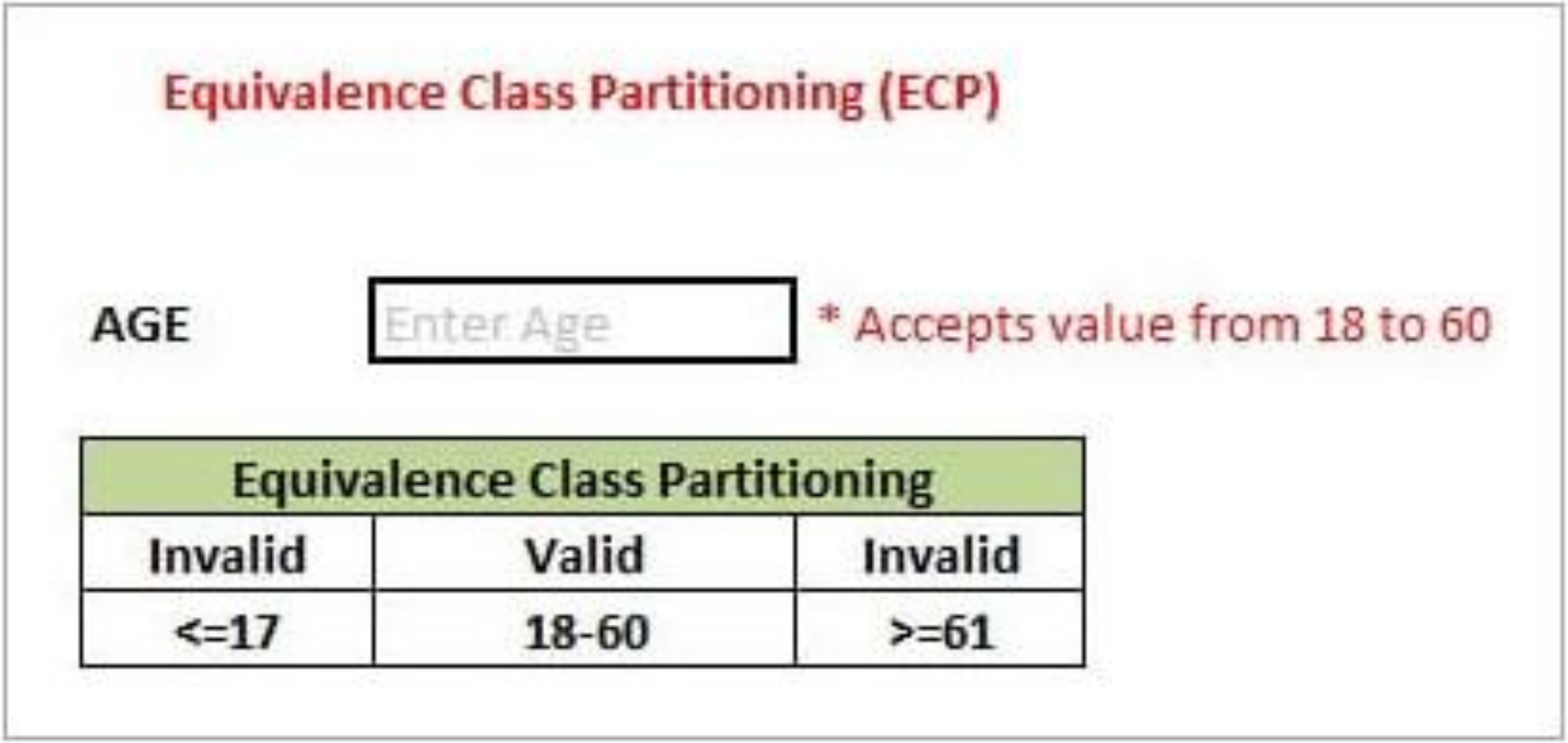
- If we want to test a field where values from 1 to 100 should be accepted, then we choose the boundary values: **1-1, 1, 1+1, 100-1, 100, and 100+1.** Instead of using all the values from 1 to 100, we just use **0, 1, 2, 99, 100, and 101**.

## 2) Equivalence Partitioning

- This technique is also known as **Equivalence Class Partitioning (ECP)**. In this technique, input values to the system or application are **divided into different classes** or groups based on its similarity in the outcome.

- Hence, instead of using each and every input value, we can now use **any one value** from the **group/class** to test the outcome. This way, we can maintain test coverage while we can **reduce the amount of rework** and most importantly the time spent.

For Example: As present in the below image, the “AGE” text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.



Equivalence Class Partitioning (ECP)

AGE | Enter Age | * Accepts value from 18 to 60

| Equivalence Class Partitioning | | |
|---|---|---|
| Invalid | Valid | Invalid |
| <=17 | 18-60 | >=61 |

- Two invalid classes will be:
- a) Less than or equal to 17.
- b) Greater than or equal to 61.

- A valid class will be anything between 18 and 60.

- We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with any one value from each set of the class is sufficient to test the above scenario.

The equivalence classes will be as follows:

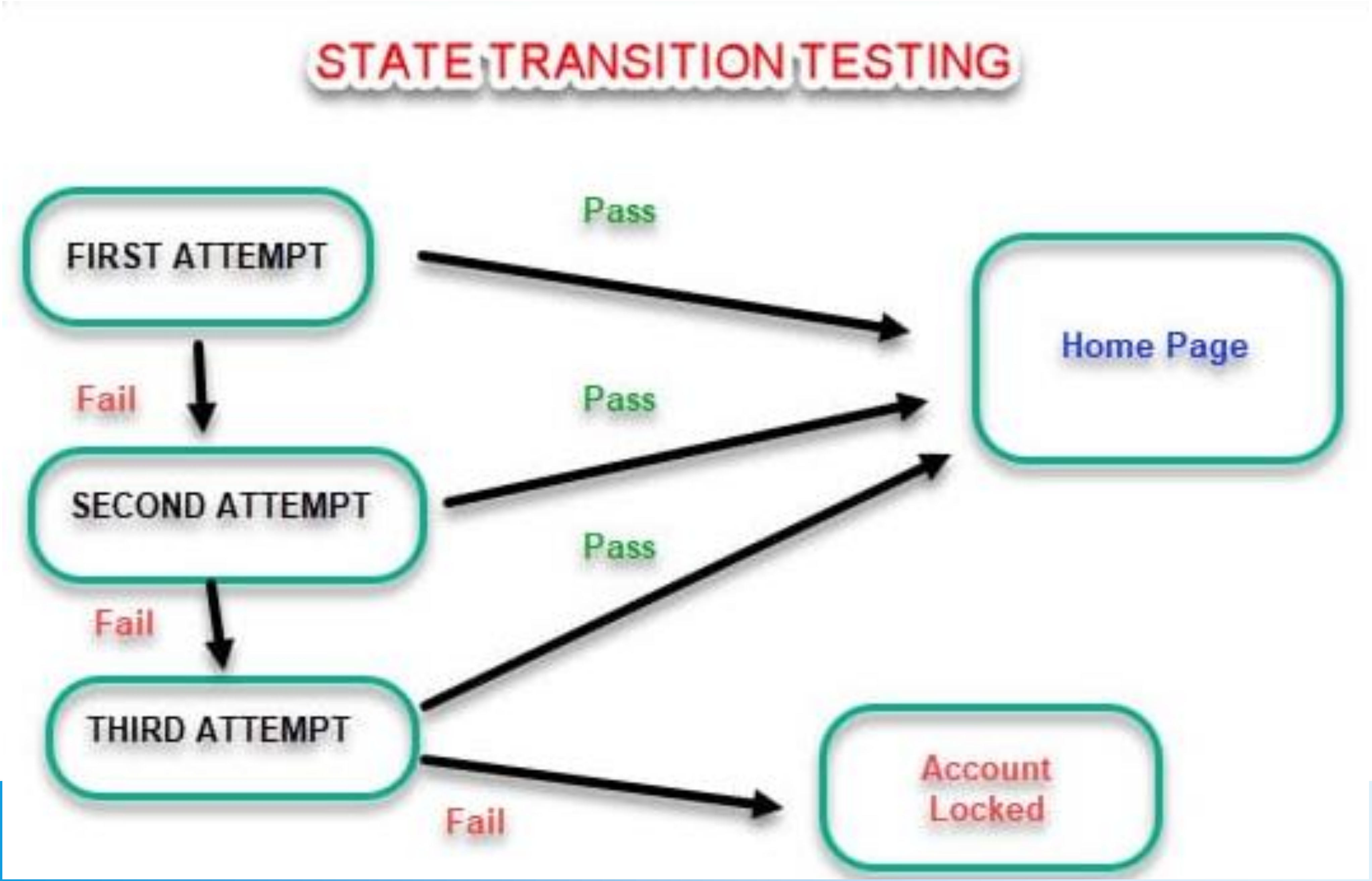
- Class I=> invalid class: < 25
- Class II => valid class:  25 t0 50
- Class III => invalid class: > 50

## State Transition Testing

- State Transition Testing is a technique that is used to test the different states of the system under test. The state of the system changes depending upon the conditions or events. The events trigger states which become scenarios and a tester needs to test them.

- A systematic state transition diagram gives a clear view of the state changes but it is effective for simpler applications. More complex projects may lead to more complex transition diagrams thereby making it less effective.

# For Example: Verification of ATM pin



## Exercise/TD

(1) Consider that, the maximum verification of an ATM pin is five (5). Assuming you were to use state transition testing to verify the maximum entering of ATM pin. Depict the graphical illustration of the scenario.

(2) In an Examination, a candidate has to score a minimum of 25 marks in order to pass the exam. The maximum that he can score is 50 marks.

(a) Identify and explain Equivalence values if the student passes the exam.

(b) Identify and explain Equivalence values for processing the student marks

(3) Below are four faulty programs. Each includes test inputs that result in failure. Answer the following questions about each program.

## Exercise/TD

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

(b) If possible, give a test case that does not execute the fault. If not, briefly explain why not.

(c) If possible, give a test case that executes the fault, but does not result in an error state.

If not, briefly explain why not.

(d) If possible give a test case that results in an error state, but not a failure. Hint: Don’t forget about the program counter. If not, briefly explain why not.

(e) For the given test case, describe the first error state. Be sure to describe the complete state

## Exercise/TD

```
/**
 * Find last index of element
 *
 * @param x array to search
 * @param y value to look for
 * @return last index of y in x; -1 if absent
 * @throws NullPointerException if x is null
 */
public int findLast (int[] x, int y)
{
   for (int i=x.length-1; i > 0; i--)
   {
      if (x[i] == y)
      {
         return i;
      }
   }
   return -1;
}
// test:  x = [2, 3, 5]; y = 2; Expected = 0
// Book website: FindLast.java
// Book website: FindLastTest.java
```

```
/**
 * Find last index of zero
 *
 * @param x array to search
 *
 * @return last index of 0 in x; -1 if absent
 * @throws NullPointerException if x is null
 */
public static int lastZero (int[] x)
{
   for (int i = 0; i < x.length; i++)
   {
      if (x[i] == 0)
      {
         return i;
      }
   }
   return -1;
}
// test:  x = [0, 1, 0]; Expected = 2
// Book website: LastZero.java
// Book website: LastZeroTest.java
```

```java
/**
 * Count positive elements
 *
 * @param x array to search
 * @return count of positive elements in x
 * @throws NullPointerException if x is null
 */
public int countPositive (int[] x)
{
   int count = 0;
   for (int i=0; i < x.length; i++)
   {
      if (x[i] >= 0)
      {
         count++;
      }
   }
   return count;
}
// test:  x = [-4, 2, 0, 2]; Expcted = 2
// Book website: CountPositive.java
// Book website: CountPositiveTest.java
```

```java
/**
 * Count odd or postive elements
 *
 * @param x array to search
 * @return count of odd/positive values in x
 * @throws NullPointerException if x is null
 */
public static int oddOrPos(int[] x)
{
   int count = 0;
   for (int i = 0; i < x.length; i++)
   {
      if (x[i]%2 == 1 || x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
// test:  x = [-3, -2, 0, 1, 4]; Expected = 3
// Book website: OddOrPos.java
// Book website: OddOrPosTest.java
```

## Exercise/TD Answer

The equivalence classes will be as follows:

- Class I=> invalid class: < 25
- Class II => valid class:  25 t0 50
- Class III => invalid class: > 50

- We need to identify Valid Equivalence values. Valid Equivalence values will be there in a Valid Equivalence class. All the values should be in Class II.

## Decision Table Testing

- We know that Black box testing involves validating the system without knowing its internal design.
- Moreover, we have also discussed that **Boundary Value Analysis** and **Equivalence Partitioning** can only handle one input condition at a time.
- So what do we do if we need to test **complex business logic** where multiple input conditions and actions are involved? Consequently, we will discuss another black box testing technique known as **Decision Table Testing**. Because this testing can handle such cases

- What is Decision Table Testing?
- How to create a Decision Table?
- Pitfalls of Decision Table Testing

## What is Decision Table Testing?

- We often test applications where the action is basis of multiple input combinations. E.g., if you are applying for a loan, then the eligibility for getting a loan will depend on several factors like age, Monthly Salary, loan amount required, etc.
- Consequently, these combinations are called **business logic** based on which the application functionality works. The decision table is a black box testing technique that is used to test these complex business logic.

- As the name itself suggests, wherever there are logical relationships like:

```
If {
(Condition = True)
then action1 ;
}
else action2;                    /*(condition = False)*/
```

- Then a tester will identify two outputs (action1 and action2) for two conditions (True and False). So based on the probable scenarios a Decision table is carved to prepare a set of test cases.

- **It is a tabular representation of input conditions and resulting actions** .i.e A decision table is a tabular representation of a decision situation, where the different combinations of alternative condition states and the resulting decisions are represented as columns in a table.
- Additionally, it shows the causes and effects. Therefore, this technique is also called a **cause-effect table.**

- Testing combinations can be a challenge, especially if the number of combinations is enormous.
- Moreover, testing **all combinations is not practically feasible** as it's not cost and time effective. Therefore, we have to be satisfied with testing just a small subset of combinations. That is to say, the success of this technique depends on our choice of combinations

## How to Create a Decision Table?

**A decision table consists of four parts:**

- The **conditions variables** are the criteria which are relevant to the decision making process. They represent the items about which information is needed to take the right decision.
- The **condition states** are the values the conditions can take: every condition has its set of condition states.
- The **actions** describe the result of a decision.
- The **action values** are the possible values for a given action

- For example: If you are to apply for a loan  in XYZ bank, then the eligibility for getting a loan will depend on several factors like age, Monthly Salary, loan amount required, etc.

## Assignment

- Team Work Presentation