

## ***Lesson 2: Black-box Testing***

---

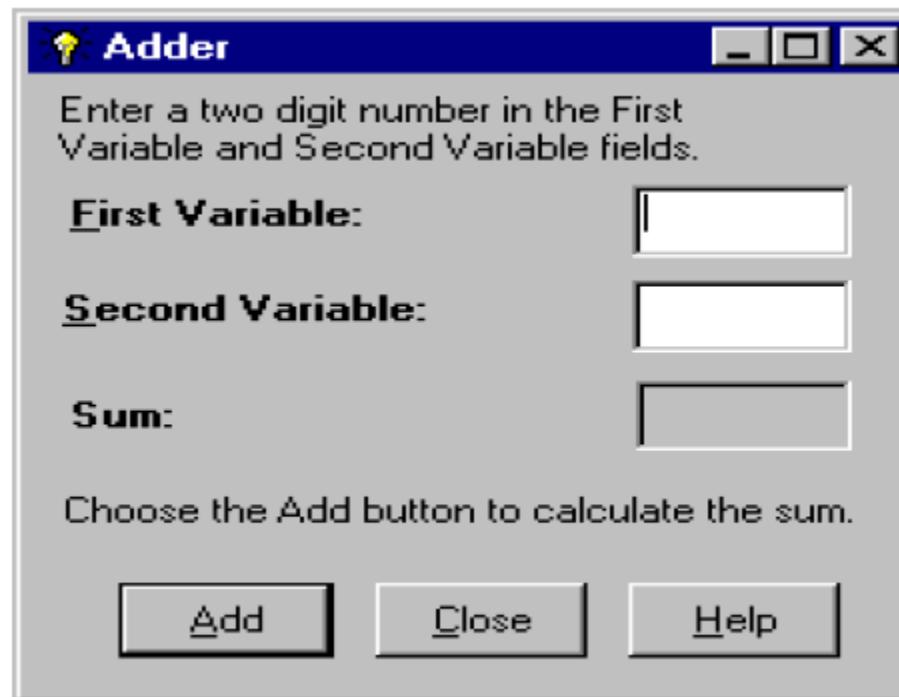
1. *Boundary Value Analysis*
2. *Equivalence Partitioning*
3. *Decision Table*
4. *State Transition*

## Example:

---

The program's specification:

- This program is designed to add two numbers, which you will enter.
- Each number should be one or two digits.



## Example:

---

The program's specification:

- This program is designed to add two numbers, which you will enter.
- Each number should be one or two digits.

- There are  $199 \times 199 = 39,601$  test cases for valid values:
  - definitely valid: 0 to 99
  - might be valid: -99 to -1
- There are infinitely many invalid cases:
  - 100 and above
  - -100 and below
  - anything non-numeric

## Black-box Test Technique

---

Black-box Testing is a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications.

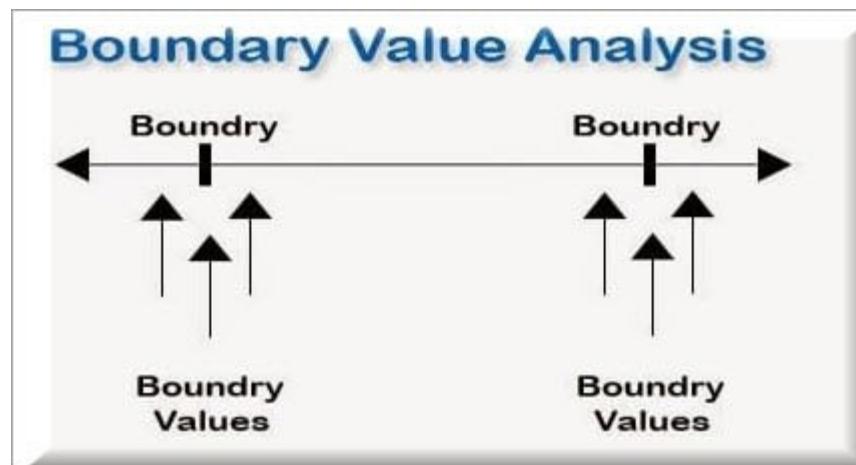
*There are 4 types of Black-box testing techniques:*

- Boundary Value Analysis
- Equivalence Partitioning
- Decision Table
- State Transition

## 2.1 Boundary Value Analysis (BVA)

---

- BVA is used to find the errors at boundaries of input domain rather than finding those errors in the centre of input. So, the basic idea in boundary value testing is to select input variable values at their: minimum (lower boundary), above the minimum, below the minimum, below the maximum, maximum and above the maximum (upper boundary).



## 2.1 Boundary Value Analysis (BVA)

---

- Suppose, if the input is the values between A and B, then we design test cases for A-1, A, A+1, and B-1, B, B+1.
- Example 1:

Age:  \* Accepts any value from 18 to 56

Use BVA technique to design test cases to check input data. How many test cases and what is the value used for testing?

## 2.1 Boundary Value Analysis (BVA)

---

- Suppose, if the input is a set of values between A and B, then design test cases for A, A+1, A-1 and B, B+1, B-1.
- Example 1:

Age:  \* Accepts any value from 18 to 56

Boundary Value Analysis		
Invalid (min-1)	Valid (min, +min, -max, max)	Invalid(max+1)
17	18,19,55,56	57

- Valid Test Cases: 18, 19, 55, 56
- Invalid Test Cases: 17, 57

## 2.1 Boundary Value Analysis (BVA)

---

- **Example 2:** Consider an example where a developer writes code for an amount text field which will accept and transfer values only from **100** to **5000**.

Amount:  Transfer

Your answer:

- Valid Test Cases: ?
- Invalid Test Cases: ?

## 2.1 Boundary Value Analysis (BVA)

---

- Example 2: Consider an example where a developer writes code for an amount text field which will accept and transfer values only from 100 to 5000.

Amount:  Transfer

Boundary Value Analysis		
Invalid (min-1)	Valid (min, +min, -max, max)	Invalid(max+1)
99	100, 101, 4999, 5000	5001

- Valid Test Cases: 100, 101, 4999, 5000
- Invalid Test Cases: 99, 5001

## 2.2 Equivalence Partitioning

---

- In this method, the input domain data is divided into different equivalence data classes – which are generally termed as ‘Valid’ and ‘Invalid’. The inputs to the software or system are divided into groups that are expected to exhibit similar behavior.
- *Example: Suppose the application you are testing accept values in the character limit of 1 – 100 only.*  
*Here, there would be three partitions: one valid partition and two invalid partitions.*
  - The valid partition: Between 1 & 100 characters.
  - The first invalid partition: 0 characters (blank)
  - The second invalid partition:  $\geq 101$  characters

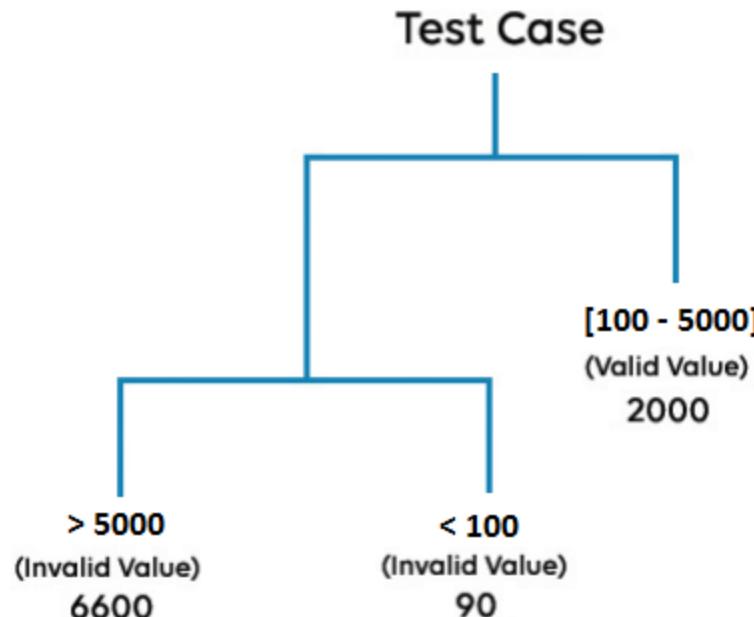
## 2.2 Equivalence Partitioning

---

**Rule 1:** If input is a range of values, then design test cases for one valid and two invalid values.

For example:

Amount:  100 – 5000 range of values



## 2.2 Equivalence Partitioning

---

**Rule 2:** If input is a set of values, then design test cases for all valid value sets and two invalid values.

**For example:** Consider any online shopping website, where every product should have a specific product ID and name. Users can search either by using name of the product or by the product ID. Here, you can consider a set of products with product IDs and you want to check for Laptops (valid value).



**Requirement says:**

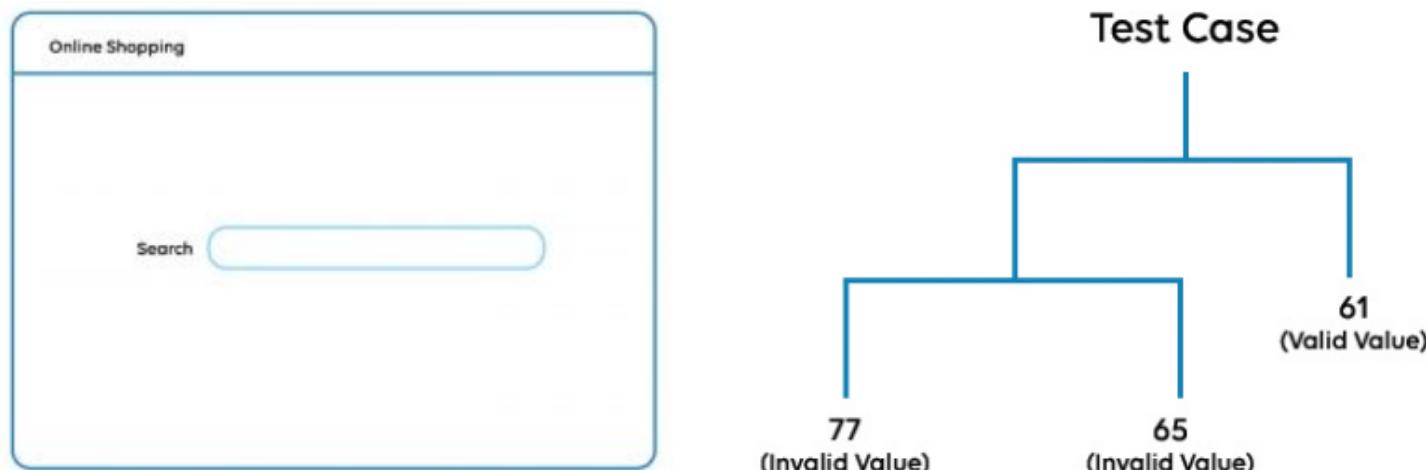
Product	Product ID
Mobiles	55
Laptops	61
Pen Drives	64
Keyboard	71
Headphones	76

## 2.2 Equivalence Partitioning

---

**Rule 2:** If input is a set of values, then design test cases for all valid value sets and two invalid values.

**For example:** Consider any online shopping website, where every product should have a specific product ID and name. Users can search either by using name of the product or by the product ID. Here, you can consider a set of products with product IDs and you want to check for Laptops (valid value).



## 2.2 Equivalence Partitioning

---

**Rule 3:** If the input is a range/set of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that invalid values are being tested for.

**For example:** How many test cases for this program?

Number:  Prime

Result: Prime

Input data: 2, 7, 13, 113, 71 (Prime); 12, 143, 112, 58, 98 (Not Prime); and 0, 1, -7, -6, -53 (Not Prime)

Does the program run well?

## 2.2 Equivalence Partitioning

---

**Rule 3:** If the input is a range of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that invalid values are being tested for.

For example:

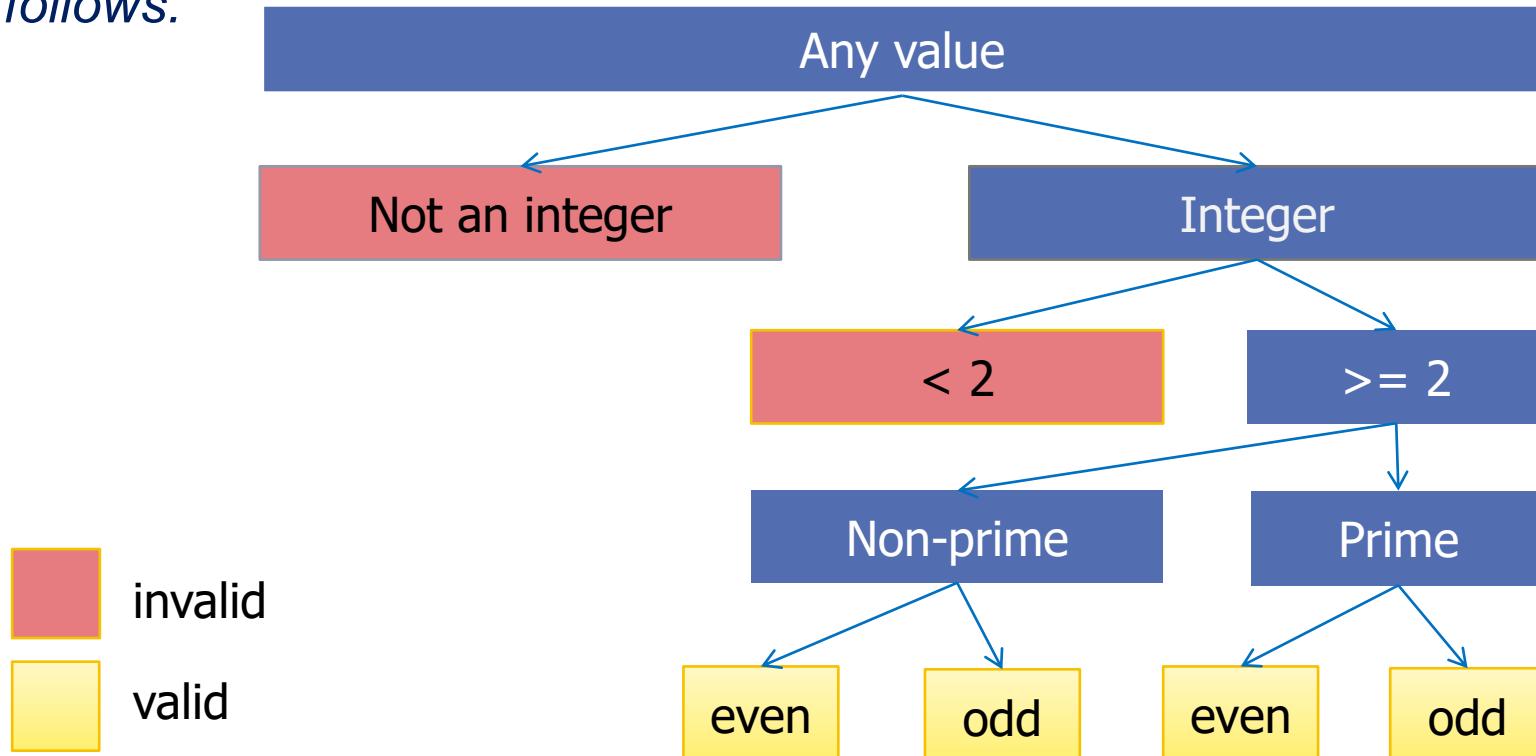
Code:

```
bool primeCheck(int n) {  
    if (n < 2)  
        return false;  
  
    for (int i = 2; i < sqrt(n); i++)  
        if (n % i == 0)  
            return false;  
  
    return true;  
}
```

## 2.2 Equivalence Partitioning

**Rule 3:** If the input is a range of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that 2 invalid values are being tested for.

For example: *If tester have more experiences, he will analyse as follows:*



## 2.2 Equivalence Partitioning

---

**Rule 3:** If the input is a range of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that 2 invalid values are being tested for.

For example:

Find the error in the following code:

```
bool primeCheck(int n) {  
    if (n < 2)  
        return false;  
  
    for (int i = 2; i <=sqrt(n); i++)  
        if (n % i == 0)  
            return false;  
  
    return true;  
}
```

## 2.2 Equivalence Partitioning

---

**Rule 3:** If the input is a range/set of values, then divide the range into equivalent parts. Then test for all the valid values and ensure that invalid values are being tested for.

For example: How many test cases for this program?

Number:  Prime

Result: Prime

Invalid Test Cases	Valid Test Cases
<p>Not an integer</p> <p>&lt; 2</p>	<p><i>Partition an even prime: 2</i></p> <p><i>Partition an odd primes: 3</i></p> <p><i>Partition an odd non-prime: 9</i></p> <p><i>Partition an even non-prime: 4</i></p>