# Black Box Testing

Ashima Tyagi

Assistant Professor

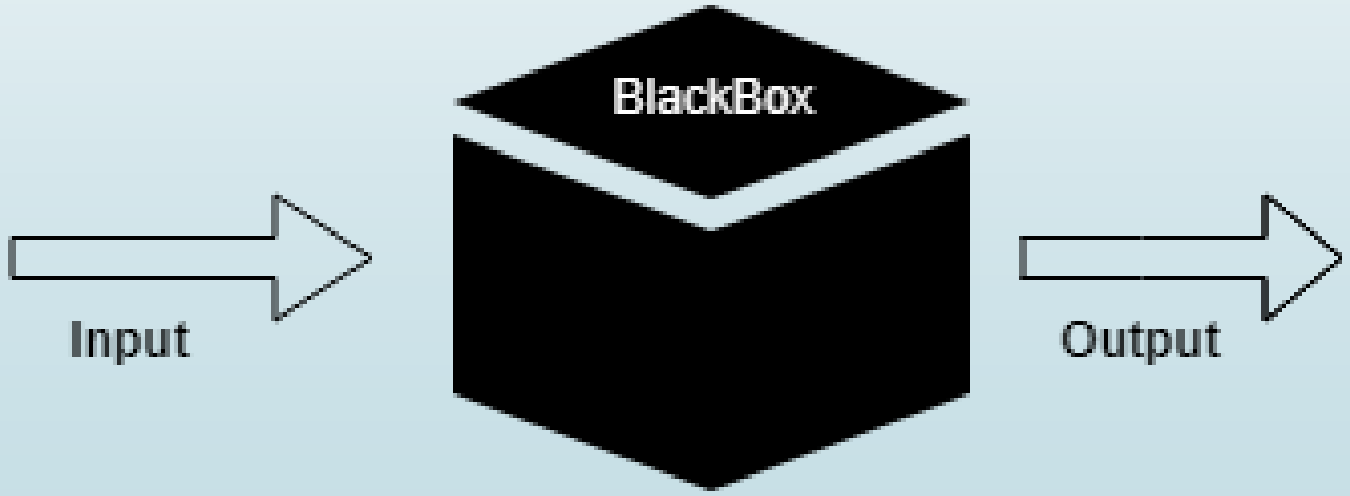
School of Computer Science & Engineering

## Outline

- Black Box Testing
- Types of Black Box Testing
- Regression Testing
- Equivalence partitioning
- Boundary value analysis
- White box Vs Black Box Testing

## Black box testing

- Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding.
- The primary source of black box testing is a specification of requirements that is stated by the customer.



## Types of Black Box Testing

Black box testing consists of following types of testing techniques:

1. Regression Testing
2. Equivalence Partitioning
3. Boundary value analysis

## 1. Regression Testing

- Regression testing is a type of software testing conducted after a code update to ensure that the update introduced no new bugs.
- This is because new code may bring in new logic that conflicts with the existing code, leading to defects.
- Usually, QA teams have a series of regression test cases for important features that they will re-execute each time these code changes occur to save time and maximize test efficiency.

- **Example:** A software team develops an e-commerce website. The testing team creates 1000 test cases to check every aspect of the application. After adding new features or updates, the QA team does regression testing by creating 100 new test cases and re-executing the previous 1000. This ensures the website works as intended and maintains software quality.

## When Can We Perform Regression Testing?

Regression testing is performed by testers in the following scenarios:

- When new functionality is added to the software application.
- When there is a requirement to change.
- When a defect is fixed.
- When a functional/performance issue is fixed.
- When there is a change in the environment.

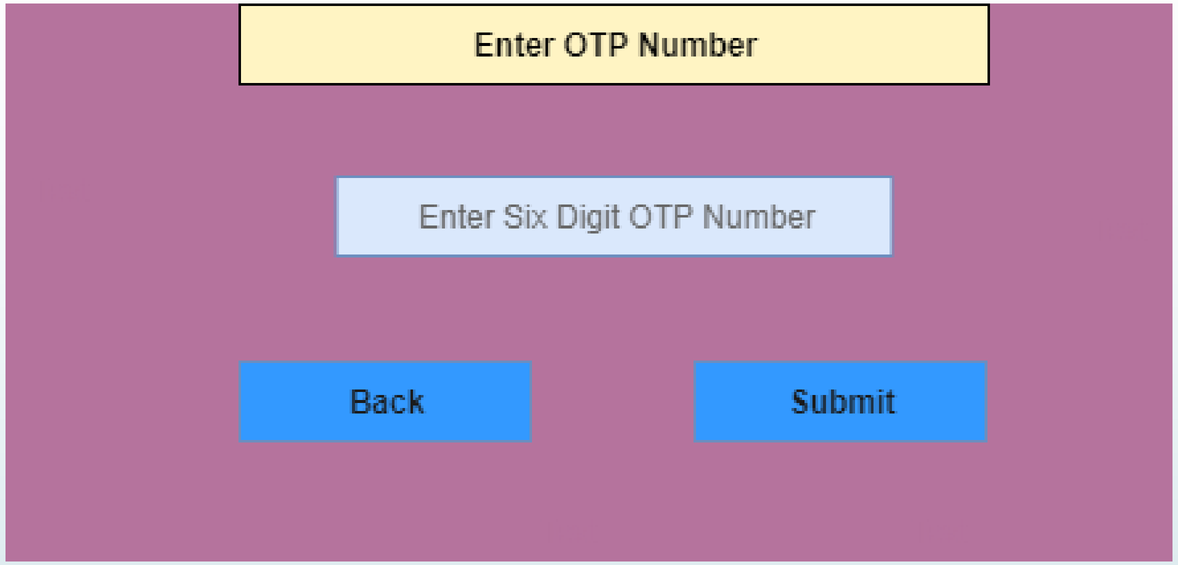
## 2. Equivalence partitioning

- Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
- If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false.
- The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once.
- Each value of every equal partition must exhibit the same behavior as other.
- The equivalence partitions are derived from requirements and specifications of the software. The advantage of this approach is, it helps to reduce the time of testing due to a smaller number of test cases from infinite to finite. It is applicable at all levels of the testing process.

## Examples of Equivalence Partitioning technique

- Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

- OTP Number = 6 digits

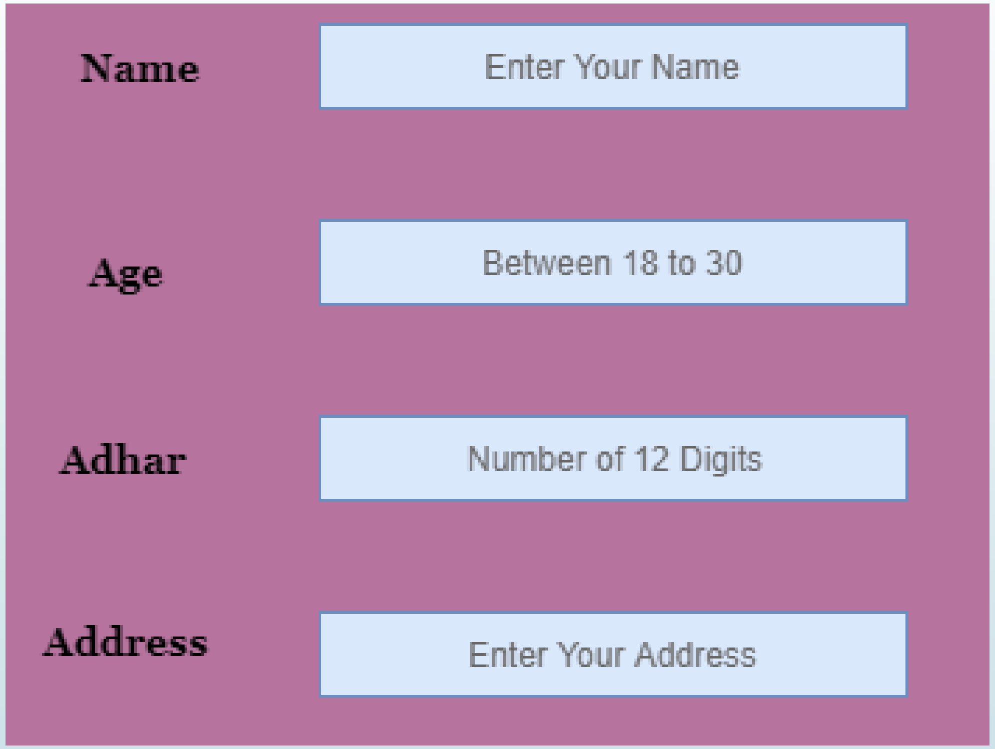


| INVALID | INVALID | VALID | VALID |
|---|---|---|---|
| 1 Test case | 2 Test case | 3 Test case | |
| DIGITS >=7 | DIGITS<=5 | DIGITS = 6 | DIGITS = 6 |
| 93847262 | 9845 | 456234 | 451483 |

## 3. Boundary Value Analysis

- BVA is used to test boundary values because the input values near the boundary have higher chances of error.
- Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.
- Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.
- The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

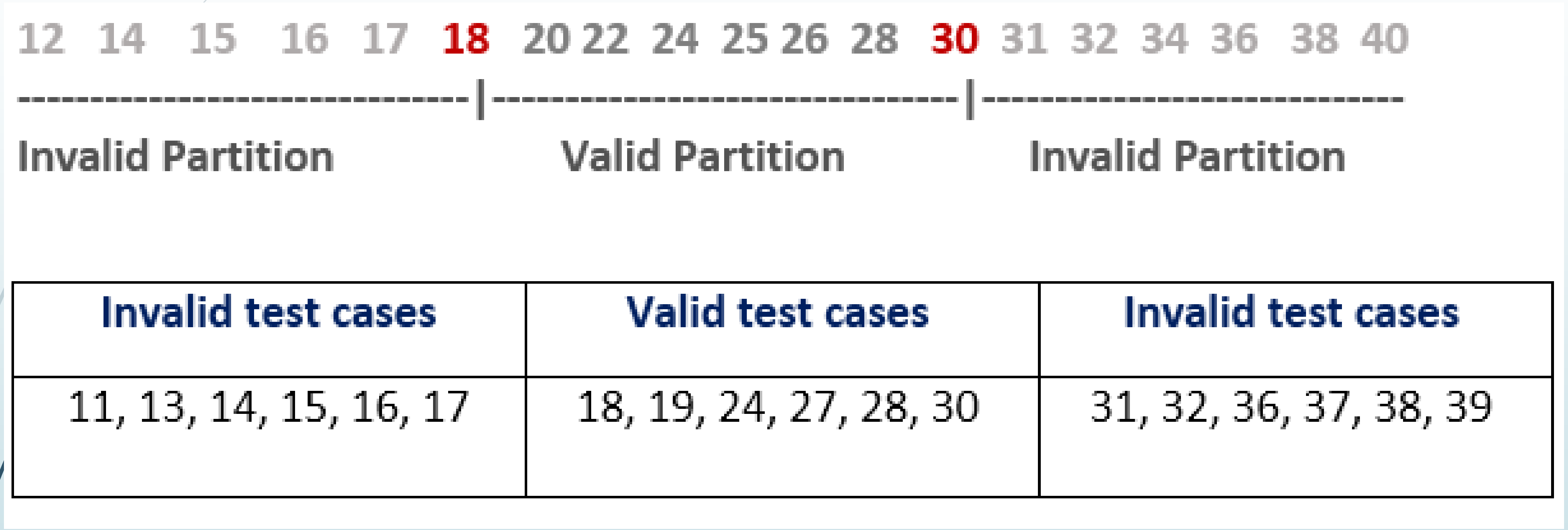


| | |
|---|---|
| Name | Enter Your Name |
| Age | Between 18 to 30 |
| Adhar | Number of 12 Digits |
| Address | Enter Your Address |

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

## Example:

- Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.



12 14 15 16 17 **18** 20 22 24 25 26 28 **30** 31 32 34 36 38 40

-----------------------------------|------------------------------------|--------------------------------

Invalid Partition Valid Partition Invalid Partition

| Invalid test cases | Valid test cases | Invalid test cases |
|---|---|---|
| 11, 13, 14, 15, 16, 17 | 18, 19, 24, 27, 28, 30 | 31, 32, 36, 37, 38, 39 |

- The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful.
- In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

- If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

# White box Vs Black Box Testing

| S.no. | On the basis of | Black Box testing | White Box testing |
|---|---|---|---|
| 1. | **Basic** | • It is a software testing technique that examines the functionality of software without knowing its internal structure or coding. | • In white-box testing, the internal structure of the software is known to the tester. |
| 2. | **Also known as** | • Black Box Testing is also known as functional testing, data-driven testing, and closed-box testing. | • It is also known as structural testing, clear box testing, code-based testing, and transparent testing. |
| 3. | **Programming knowledge** | • In black-box testing, there is less programming knowledge is required. | • In white-box testing, there is a requirement of programming knowledge. |
| 4. | **Algorithm testing** | • It is not well suitable for algorithm testing. | • It is well suitable and recommended for algorithm testing. |
| 5. | **Usage** | • It is done at higher levels of testing that are system testing and acceptance testing. | • It is done at lower levels of testing that are unit testing and integration testing. |
| 6. | **Automation** | • It is hard to automate black-box testing due to the dependency of testers and programmers on each other. | • It is easy to automate the white box testing. |
| 7. | **Tested by** | • It is mainly performed by the software testers. | • It is mainly performed by developers. |
| 8. | **Time-consuming** | • It is less time-consuming. In Black box testing, time consumption depends upon the availability of the functional specifications. | • It is more time-consuming. It takes a long time to design test cases due to lengthy code. |
| 9. | **Defect detection** | • In black-box testing, defects are identified once the code is ready. | • Whereas, in white box testing, there is a possibility of early detection of defects. |
| 10. | **Errors** | • It does not find the errors related to the code. | • In white-box testing, there is the detection of |

# Thank You