## Software Testing

**Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is[Defect](https://www.guru99.com/defect-management-process.html)free.

**Error :-A mistake in coding is called Error.**

**Defect :-error found by tester is called Defect also called bug.**

**Bug :-Tester found a any defect error is called a bug.**

**Fault :-**  **Faults or bugs in a hardware or software may cause errors.**

**Failure:- bug and fault find then they called a failure.**

**Bug life cycle**

Bug life cycle, also known as defect Life cycle.bug life cycle start with a unintentional software bug/behvaiour and end when the assigned develpor fixed bug.

A bug when found should be communicated and assigned to adeveloer that can fix it.

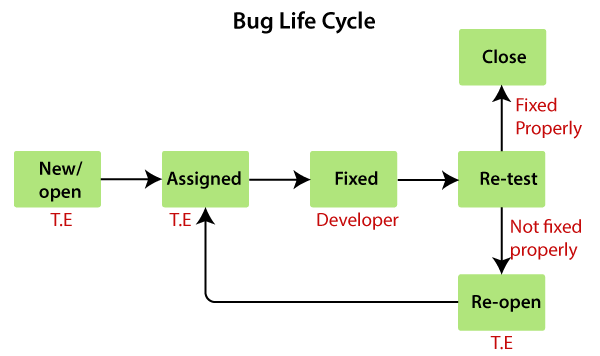


Fig:-Bug life cycle.

Defect Life Cycle States:

* **New – tester finds a any defect is called new/open.**
* **Assigned –**assigned the teamwise work.
* **Active -**The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
* **Test -**The Defect is fixed and ready for testing.
* **Verified -**The Defect that is retested and the test has been verified by QA.
* **Closed -**The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
* **Reopened -**When the defect is NOT fixed, QA reopens/reactivates the defect.
* **Deferred -**When a defect cannot be addressed in that particular cycle it is deferred to future release.
* **Rejected -**A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

**LEVEL OF TESTING:-**

**1.Unit testing**

**2.Integration Testing**

**3.syetem testing**

**4.accepatnece testing**

**Software testing type:-**

**1.Manual Testing**

**2.Automation Testing**

**1.Manual Testing:-**

**1.white -box testing**

**2.black box testing**

**3.greay box testing**

**1.white box testing:-**

**Checking internally code.**

## White Box Techniques

1. [**Data Flow Testing**](https://www.javatpoint.com/data-flow-testing-in-white-box-testing)

**Checking flow of data .**

**Example:**

**1.** read x, y;

**2.** if(x>y)

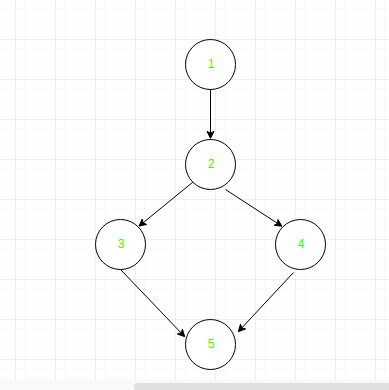
**3.** a = x+1

else

**4.** a = y-1

**5.** print a;

**Control flow graph of above example:**

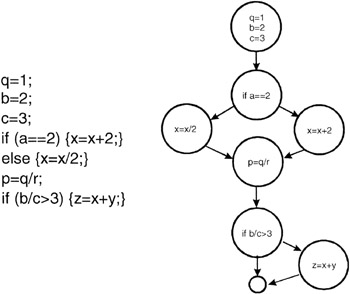


| Variable | Defined at node | Used at node |
| --- | --- | --- |
| X | 1 | 2, 3 |
| Y | 1 | 2, 4 |
| A | 3, 4 | 5 |

1. [**Control Flow Testing**](https://www.javatpoint.com/control-flow-testing-in-white-box-testing)

**Main aim to control flow testing is there are checking control.**

The following code example is represented by its associated flow graph:



1. [**Branch Coverage Testing**](https://www.javatpoint.com/branch-coverage-testing-in-white-box-testing)

**How many branch coverage to it.**

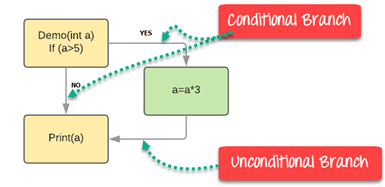
Demo(int a) {

If (a> 5)

a=a\*3

Print (a)

}



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** | **Branch Coverage** |
| 1 | 2 | 2 | 50% | **33%** |
| 2 | 6 | 18 | 50% | **67%** |

1. [**Statement Coverage Testing**](https://www.javatpoint.com/statement-coverage-testing-in-white-box-testing)

**How many number of line to excuted.**

Prints (int a, int b) {

int result = a+ b;

If (result> 0)

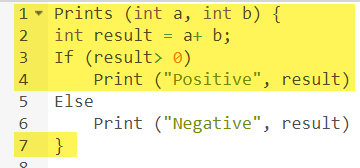
Print ("Positive", result)

Else

Print ("Negative", result)

}

If A = 3, B = 9



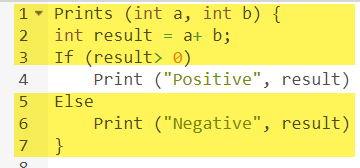
The statements marked in yellow color are those which are executed as per the scenario

Number of executed statements = 5, Total number of statements = 7

Statement Coverage: 5/7 = 71%

**Scenario 2:**

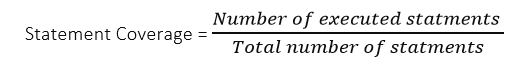
If A = -3, B = -9



The statements marked in yellow color are those which are executed as per the scenario.

Number of executed statements = 6

Total number of statements = 7



Statement Coverage: 6/7 = 85%

1. [**Decision Coverage Testing**](https://www.javatpoint.com/decision-coverage-testing-in-white-box-testing)

**To check decision wise testing.**

Demo(int a) {

If (a> 5)

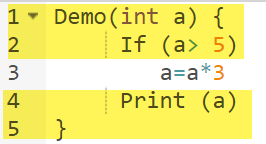
a=a\*3

Print (a)

}

**Scenario 1:**

Value of a is 2

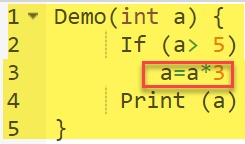


The code highlighted in yellow will be executed. Here the “No” outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

**Scenario 2:**

Value of a is 6



The code highlighted in yellow will be executed. Here the “Yes” outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** |
| 1 | 2 | 2 | 50% |
| 2 | 6 | 18 | 50% |

## 2. Black box Testing:-

## Checking External behviour of testing.

**Black box techniques:-**

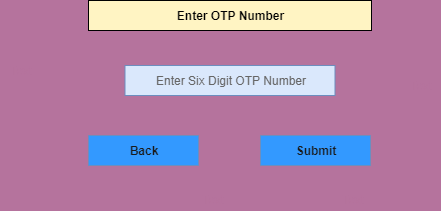
### **Equivalence Partitioning**

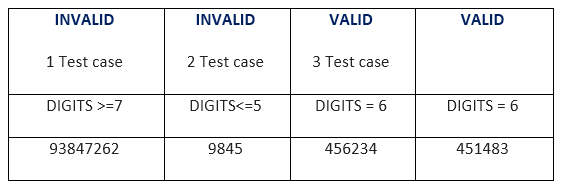
1. **Boundary value analysis**
2. [**Decision Table**](https://www.javatpoint.com/decision-table-technique-in-black-box-testing)
3. [**State Transition**](https://www.javatpoint.com/state-transition-technique-in-black-box-testing)
4. **Error Gusessing**
5. **Graph -Based techquies**
6. **comparison testing**

### **1.Equivalence Partitioning**

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.

1. 1. OTP Number = 6 digits





**2.Boundary value analysis:-**

In boundary value analysis both valid and invalid input are being tested to verify.

For Example – A person give a loan 40,000

But boundary value is 25,000 to 30,000

So basically checking two way like left and right value

For 25000 is

24,999

25,000

25,500

For 30,000 is

29,999

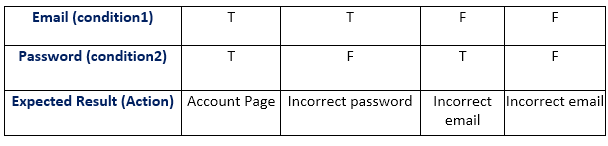
30,000

30,000

**3.**[**Decision Table**](https://www.javatpoint.com/decision-table-technique-in-black-box-testing)

This technique is related to the correct combination of inputs and determines the result of various combinations of input.

Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.



In the table, there are four conditions or test cases to test the login function. In the first condition if both email and password are correct, then the user should be directed to account's Homepage.

In the second condition if the email is correct, but the password is incorrect then the function should display Incorrect Password. In the third condition if the email is incorrect, but the password is correct, then it should display Incorrect Email.

Now, in fourth and last condition both email and password are incorrect then the function should display Incorrect Email.

In order to find the number of all possible conditions, tester uses 2n formula where n denotes the number of inputs; in the example there is the number of inputs is 2 (one is true and second is false).

Number of possible conditions = 2^ Number of Values of the second condition  
Number of possible conditions =2^2 = 4

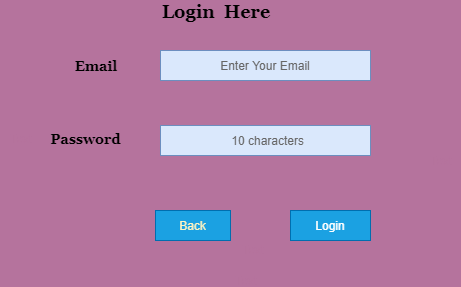
While using the decision table technique, a tester determines the expected output, if the function produces expected output, then it is passed in testing, and if not then it is failed. Failed software is sent back to the development team to fix the defect.

**4.State transition:-**

It is used to capture the behavior of the software application when different input values are given to the same function.

We all use the ATMs, when we withdraw money from it, it displays account details at last. Now we again do another transaction, then it again displays account details, but the details displayed after the second transaction are different from the first transaction, but both details are displayed by using the same function of the ATM. So the same function was used here but each time the output was different, this is called state transition. In the case of testing of a software application, this method tests whether the function is following state transition specifications on entering different inputs.

This applies to those types of applications that provide the specific number of attempts to access the application such as the login function of an application which gets locked after the specified number of incorrect attempts. Let's see in detail, in the login function we use email and password, it gives a specific number of attempts to access the application, after crossing the maximum number of attempts it gets locked with an error message.



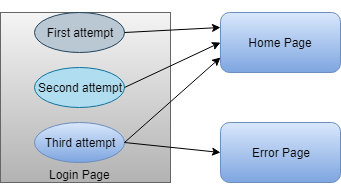
**Let see in the diagram:**

47.5M

803

History of Java

There is a login function of an application which provides a maximum three number of attempts, and after exceeding three attempts, it will be directed to an error page.



**State transition table**

|  |  |  |  |
| --- | --- | --- | --- |
| **STATE** | **LOGIN** | **VALIDATION** | **REDIRECTED** |
| S1 | First Attempt | Invalid | S2 |
| S2 | Second Attempt | Invalid | S3 |
| S3 | Third Attempt | Invalid | S5 |
| S4 | Home Page |  |  |
| S5 | Error Page |  |  |

In the above state transition table, we see that state S1 denotes first login attempt. When the first attempt is invalid, the user will be directed to the second attempt (state S2). If the second attempt is also invalid, then the user will be directed to the third attempt (state S3). Now if the third and last attempt is invalid, then the user will be directed to the error page (state S5).

But if the third attempt is valid, then it will be directed to the homepage (state S4).

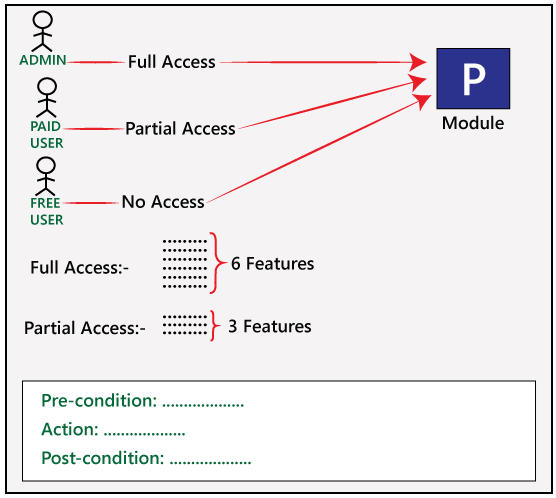
**Let's see state transition table if third attempt is valid:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STATE** | **LOGIN** | **VALIDATION** | **REDIRECTED** |
| S1 | First Attempt | Invalid | S2 |
| S2 | Second Attempt | Invalid | S3 |
| S3 | Third Attempt | Valid | S4 |
| S4 | Home Page |  |  |
| S5 | Error Page |  |  |

By using the above state transition table we can perform testing of any software application. We can make a state transition table by determining desired output, and then exercise the software system to examine whether it is giving desired output or not.

The use case is functional testing of the black box testing used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

It is a graphic demonstration of business needs, which describe how the end-user will cooperate with the software or the application. The use cases provide us all the possible techniques of how the end-user uses the application as we can see in the below image, that how the **use case** will look like:



In the above image, we can see that a sample of a use case where we have a requirement related to the customer requirement specification (CRS).

For **module P** of the software, we have six different features.

Play Video

Play

Unmute

Current TimeÂ 0:24

/

DurationÂ 4:57

Loaded: 100.00%

Â

Fullscreen

x[[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack" \t "_blank)

And here, **Admin** has access to all the **six features**, the **Paid user** has access to the **three features** and for the **Free user**, there is **no access** provided to any of the features.

Like for **Admin**, the different conditions would be as below:

**Pre-condition**→ Admin must be generated

**Action**→ Login as Paid user

**Post-condition**→ 3 features must be present

And for **Free user**, the different condition would be as below:

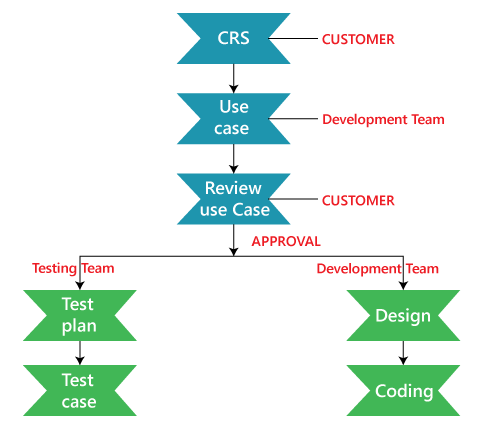
**Pre-condition**→ free user must be generated

**Action**→ Login as a free user

**Post-condition**→ no features

## Who writes the use case?

The client provides the customer requirement specification for the application, then the development team will write the **use case** according to the CRS, and the use case is sent to the customer for their review.



If the client approves it, then the approved **use case** is sent to the development team for further design and coding process and these approved use case is also sent to the testing team, so they can start writing the test plan and later on start writing the test cases for the different features of the software.

In the below scenario, there is a tester who represents the user to use the functions of a system one by one. In this scenario, there is an actor who represents the user to use the functions of a software system.

This describes step-by-step functionality of the software application which can be understood with an example, assume that there is a software application of online money transfer. The various steps for transferring money are as follows:

* The user does login for the authentication of the actual user.
* The system checks ID and password with the database to ensure that whether it is a valid user or not.
* If the verification succeeds, the server connects the user to the account page, otherwise returns to the login page.
* In the account page, there are several options because the examiner is checking the money transfer option; the user goes into the money transfer option.
* After successful completion of this step, the user enters the account number in which he wants to transfer money. The user also need to enter other details like bank name, amount, IFSC code, home branch, etc.

In the last step, if there is a security feature that includes verification of the ATM card number and PIN, then enter the ATM card number, PIN and other required details.

If the system is successfully following all the steps, then there is no need to design test cases for this function. By describing the steps to use, it is easy to design test cases for software systems.

**5.Error Gusessing**

Purpose of Error guessing

The main purpose of the error guessing technique is to deal with all possible errors which cannot be identified as informal testing.

* The main purpose of error guessing technique is to deal with all possible errors which cannot be identified informal testing.

The main purpose of this technique is to identify common errors at any level of testing by exercising the following tasks:

* Enter blank space into the text fields.
* Null pointer exception.
* Enter invalid parameters.
* Divide by zero.
* Use maximum limit of files to be uploaded.
* Check buttons without entering values.

**6.Graph -Based techquies**

**. Graph Based Testing is also called as State Based Testing**

ii. It provides a framework for model based testing.

iii. Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph.

iv. Transaction flow testing – nodes represent steps in some transaction and links represent logical connections between steps that need to be validated.

v. Finite state modeling – nodes represent user observable states of the software and links represent transitions between states.

vi. Data flow modeling – nodes are data objects and links are transformations from one data object to another.

vii. Timing modeling – nodes are program objects and links are sequential connections between these objects, link weights are required execution times.

**Steps in graph testing:**

i. Build a graph model.

ii. Identify the test requirements.

iii. Select test paths to cover those requirements.

In order to design test design cases following steps are used:

i. Understanding the system

ii. Identifying states, inputs and guards.

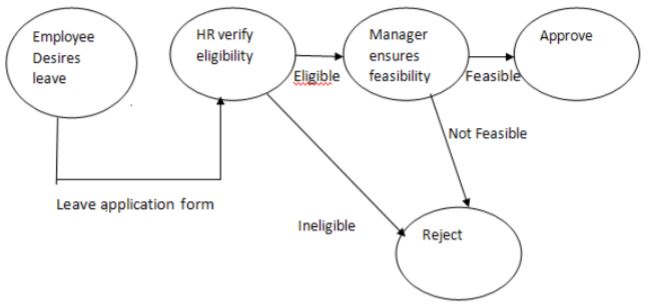
iii. Create a state graph model of the application.

iv. Verify whether State graph that we modeled is correct in all details.

v. Generate sequence of test actions.

**Derive test data so that those test paths can be executed.**

**Example:**



**7.comparison testing**

Comparison testing comprises of comparing the contents of files, databases, against actual results. They are capable of highlighting the differences between expected and actual results.

* **Verfication and validation:-**

| Verification | Validation |
| --- | --- |
| It includes checking documents, design, codes and programs. | It includes testing and validating the actual product. |
| Verification is the static testing. | Validation is the dynamic testing. |
| It does *not* include the execution of the code. | It includes the execution of the code. |
| Methods used in verification are reviews, walkthroughs, inspections and desk-checking. | Methods used in validation are Black Box Testing, White Box Testing and non-functional testing. |
| It checks whether the software conforms to specifications or not. | It checks whether the software meets the requirements and expectations of a customer or not. |
| It can find the bugs in the early stage of the development. | It can only find the bugs that could not be found by the verification process. |
| The goal of verification is application and software architecture and specification. | The goal of validation is an actual product. |
| Quality assurance team does verification. | Validation is executed on software code with the help of testing team. |
| It comes before validation. | It comes after verification. |
| It consists of checking of documents/files and is performed by human. | It consists of execution of program and is performed by computer. |

**Differences between Black Box Testing vs White Box Testing:**

|  | Black Box Testing | White Box Testing |
| --- | --- | --- |
| 1. | It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| 2. | Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| 3. | It is mostly done by software testers. | It is mostly done by software developers. |
| 4. | No knowledge of implementation is needed. | Knowledge of implementation is required. |
| 5. | It can be referred to as outer or external software testing. | It is the inner or the internal software testing. |
| 6. | It is a functional test of the software. | It is a structural test of the software. |
| 7. | This testing can be initiated based on the requirement specifications document. | This type of testing of software is started after a detail design document. |
| 8. | No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| 9. | It is the behavior testing of the software. | It is the logic testing of the software. |
| 10. | It is applicable to the higher levels of testing of software. | It is generally applicable to the lower levels of software testing. |
| 11. | It is also called closed testing. | It is also called as clear box testing. |
| 12. | It is least time consuming. | It is most time consuming. |
| 13. | It is not suitable or preferred for algorithm testing. | It is suitable for algorithm testing. |
| 14. | Can be done by trial and error ways and methods. | Data domains along with inner or internal boundaries can be better tested. |
| 15. | **Example:** Search something on google by using keywords | **Example:** By input to check and verify loops |
|  | **Black-box test design techniques-**   * Equlivalnce Partiniong * Boundary Value anaylsis * Decision table testing * State transtion * Error guessing * Graph Based testing method * Comparison testing | **White-box test design techniques-**   * Data flow testing * Control flow Testing * Branch Coverage testing * Statement coverarge * Decision coverage |
| 17. | **16.Types of Black Box Testing:**   * Functional Testing * Non-functional testing * Regression Testing | **Types of White Box Testing:**   * Path Testing * Loop Testing * Condition testing |
| 18. | It is less exhaustive as compared to white box testing. | It is comparatively more exhaustive than black box testing. |