Chapter No 6: Software Testing and Maintenance

6.1 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 free**. It involves execution of software/system components **using manual or automated tools** to evaluate one or more properties of interest. The purpose of software testing is **to identify errors, gaps or missing requirements** in contrast to actual requirements.

Software Testing is a process of executing program with an intention of finding errors.

Why Software Testing is Important?

Software testing is Important because if there are any bugs or errors in the software, **it can be identified early and can be solved before delivery of the software product.**

Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Testing in Software Engineering

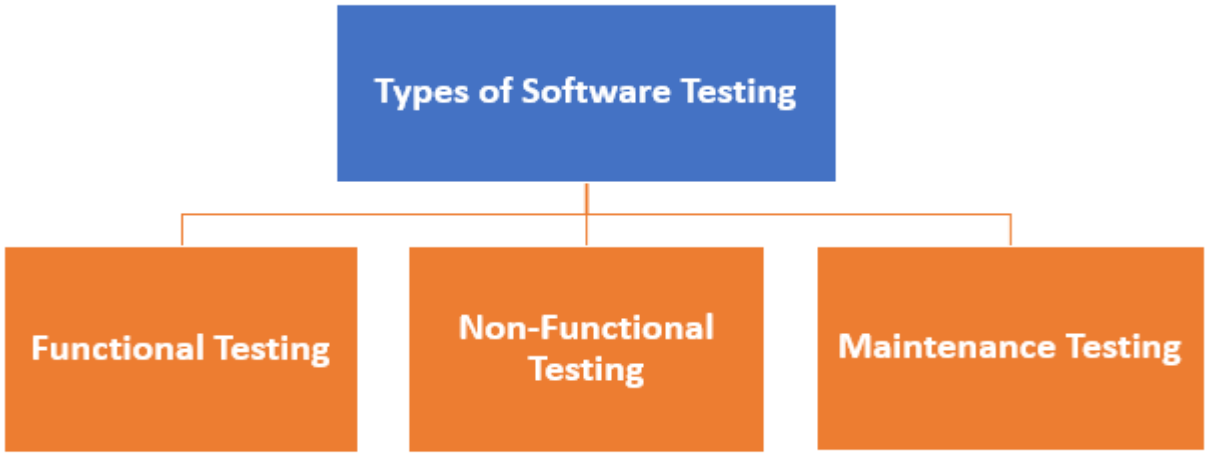
As per ANSI/IEEE 1059, **Testing in Software Engineering is a process of evaluating a software product to find whether the current software product meets the required conditions or not.** The testing process involves evaluating the features of the software product for requirements in terms of any **missing requirements, bugs or errors, security, reliability and performance.**

- Software testing is defined as an activity **to check whether the actual results match the expected results and to ensure that the software system is Defect free.**
- Testing is important **because software bugs could be expensive or even dangerous.**
- The important are reasons for using software testing are: **cost-effective, security, product quality, and customer satisfaction.**
- Typically Testing is classified into **three categories** **functional testing, non-functional testing or performance testing, and maintenance.**
- The important strategies in software engineering are: **unit testing, integration testing, system testing , validation testing (User acceptance testing).**

➢ Types of Software Testing

Typically Testing is classified into three categories.

- **Functional Testing**
- **Non-Functional Testing or Performance Testing**
- **Maintenance (Regression and Maintenance)**



➢ Testing Strategies (levels) in Software Engineering

Here are important testing strategies in software engineering:

Unit Testing: This software testing approach is followed by the programmer to test the unit of the program. It helps developers to know whether the individual unit of the code is working properly or not.

Integration testing: It focuses on the construction and design of the software. You need to see that the integrated units are working without errors or not.

System testing: In this method, your software is compiled as a whole and then tested as a whole. This testing strategy checks the functionality, security, portability, amongst others.

User Acceptance Testing (Validation Testing):

6.2 Types of Software Testing

1. Functional Testing

2. Non-Functional Testing

3. Maintenance

1. What is Functional Testing?

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

Functional testing mainly involves black box testing and **it is not concerned about the source code of the application**. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application under Test. **The testing can be done either manually or using automation**.

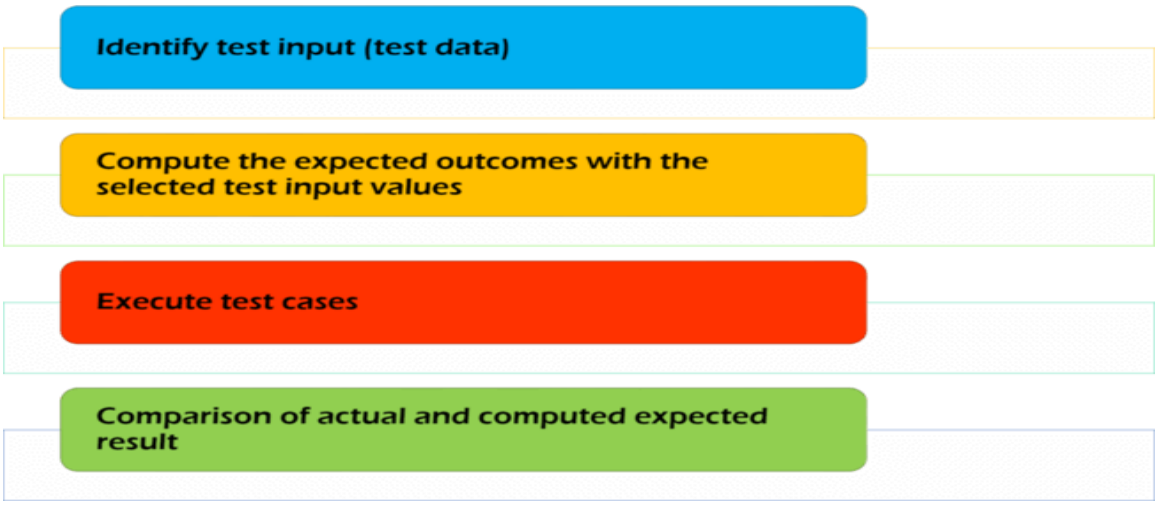
What do you test in Functional Testing?

The prime objective of Functional testing is **checking the functionalities of the software system.** It mainly concentrates on -

- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

How to do Functional Testing

- Understand the Functional Requirements
- Identify test input or test data based on requirements
- Compute the expected outcomes with selected test input values
- Execute test cases
- Compare actual and computed expected results



Functional Vs Non-Functional Testing:

Functional Testing	Non-Functional Testing
Functional testing is performed using **the functional specification provided by the client and verifies the system against the functional requirements.**	Non-Functional testing checks **the Performance, reliability, scalability and other non-functional aspects of the software system**.
Functional testing is executed first	Non-functional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing.
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing

Examples of Functional testing are	Examples of Non-functional testing are
Unit Testing Smoke Testing Sanity Testing Integration Testing White box testing Black Box testing User Acceptance testing Regression Testing	Performance Testing Load Testing Volume Testing Stress Testing Security Testing Installation Testing Penetration Testing Compatibility Testing Migration Testing

Functional Testing Tools

Here is a list of popular **Functional Testing Tools**. They are explained as follows:

- Selenium - Popular Open Source Functional Testing Tool
- QTP - Very user-friendly Functional Test tool by HP
- JUnit- Used mainly for Java applications and this can be used in Unit and System Testing
- soapUI - This is an open source functional testing tool, mainly used for Web service testing. It supports multiple protocols such as HTTP, SOAP, and JDBC.
- Watir - This is a functional testing tool for web applications. It supports tests executed at the web browser and uses a ruby scripting language.

6.3 Types of Functional Testing

White-Box Testing

1. Basic Path Testing
2. Control structure Testing
   a. Branch Testing
   b. Condition Testing
   c. Data Flow Testing
   d. Loop Testing

Black box testing

1. Equivalence Class Testing (Equivalence Partitioning)
2. Boundary Value Testing

UAT (User Acceptance Testing)

What is UAT (User Acceptance Testing)?

- User Acceptance Testing (UAT) **is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment.**
- UAT is done in the final phase of testing after functional, integration and system testing is done.
- **Purpose of UAT:** The main **Purpose of UAT** is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing.
- User Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is kind of black box testing where two or more end-users will be involved.

- **Who Performs UAT?**

  - ✓ Client
  - ✓ End users

- **Need of User Acceptance Testing**

Need of User Acceptance Testing arises once software has undergone Unit, Integration and System testing because developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them, so for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.

- Developers code software based on requirements document which is their "own" understanding of the requirements and **may not actually be what the client needs from the software**.
- Requirements changes during the course of the project may not be communicated effectively to the developers.

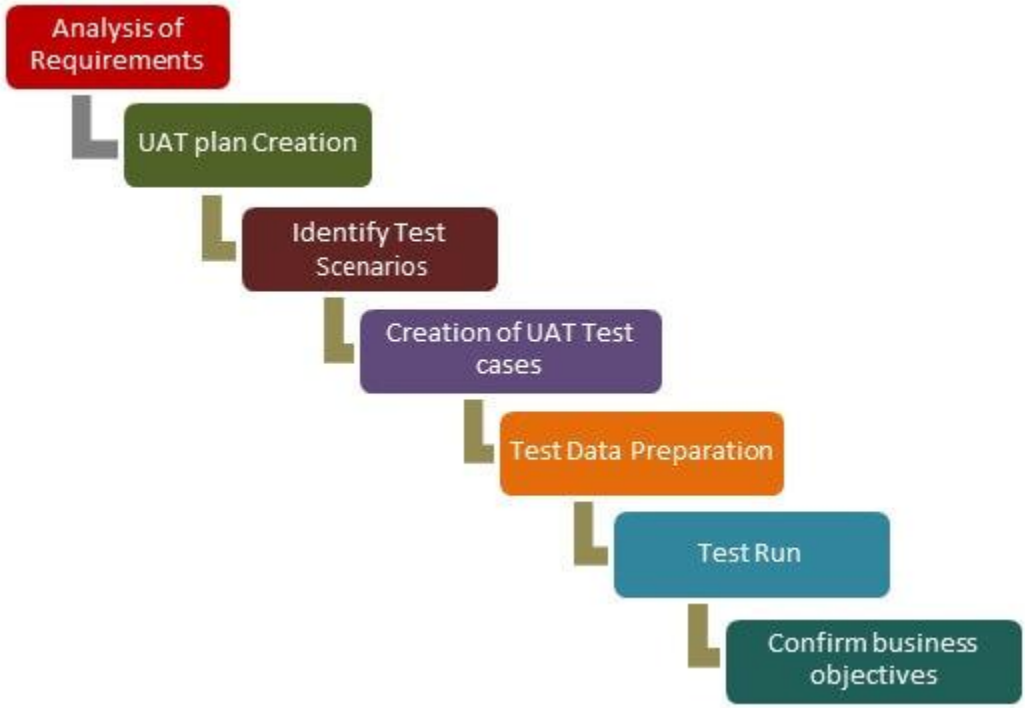
- **Prerequisites of User Acceptance Testing:**

Following are the entry criteria for User Acceptance Testing:

- ✓ Business Requirements must be available.
- ✓ Application Code should be fully developed
- ✓ Unit Testing, Integration Testing & System Testing should be completed
- ✓ No Showstoppers, High, Medium defects in System Integration Test Phase -
- ✓ Only Cosmetic error is acceptable before UAT
- ✓ Regression Testing should be completed with no major defects
- ✓ All the reported defects should be fixed and tested before UAT
- ✓ Traceability matrix for all testing should be completed
- ✓ UAT Environment must be ready
- ✓ Sign off mail or communication from System Testing Team that the system is ready for UAT execution

- **How to do UAT Testing**

UAT is done by the intended users of the system or software. This type of Software Testing usually happens at the client location which is known as Beta Testing. Once Entry criteria for UAT are satisfied, following are the tasks need to be performed by the testers:



- **UAT Process**

  - ✓ Analysis of Business Requirements
  - ✓ Creation of UAT test plan
  - ✓ Identify Test Scenarios
  - ✓ Create UAT Test Cases
  - ✓ Preparation of Test Data(Production like Data)
  - ✓ Run the Test cases
  - ✓ Record the Results
  - ✓ Confirm business objectives

Step 1) Analysis of Business Requirements

One of the most important activities in the UAT is to identify and develop test scenarios. These test scenarios are derived from the following documents:

- Project Charter
- Business Use Cases
- Process Flow Diagrams

- Business Requirements Document(BRD)
- System Requirements Specification(SRS)

Step 2) Creation of UAT Plan:

The UAT test plan outlines the strategy that will be used to verify and ensure an application meets its business requirements. It documents entry and **exit criteria for UAT, Test scenarios and test cases approach and timelines of testing**.

Step 3) Identify Test Scenarios and Test Cases:

Identify the test scenarios with respect to high-level business process and create test cases with clear test steps. Test Cases should sufficiently cover most of the UAT scenarios. Business Use cases are input for creating the test cases.

Step 4) Preparation of Test Data:

It is best advised to use live data for UAT. Data should be scrambled for privacy and security reasons. Tester should be familiar with the database flow.

Step 5) Run and record the results:

Execute test cases and report bugs if any. Re-test bugs once fixed. Test Management tools can be used for execution.

Step 6) Confirm Business Objectives met:

Business Analysts or UAT Testers needs to send a sign off mail after the UAT testing. After sign-off, the product is good to go for production. Deliverables for UAT testing are Test Plan, UAT Scenarios and Test Cases, Test Results and Defect Log

- **Exit criteria for UAT:**

Before moving into production, following needs to be considered:

- No critical defects open
- Business process works satisfactorily
- UAT Sign off meeting with all stakeholders

Qualities of UAT Testers:

UAT Tester should possess good knowledge of the business. He should be independent and think as an **unknown user to the system**. Tester should be Analytical and Lateral thinker and combine all sort of data to make the UAT successful.

White Box Vs Black Box Testing:

Black Box Testing	White Box Testing
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.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.

Black Box Testing	White Box Testing
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.

White-box testing:

- Knowing the internal workings of a product, **test that all internal operations are performed according to specifications** and all internal components have been exercised.
- **Types of White Box Testing:**

  1) Basic Path Testing

  2) Control Structure Testing

1) Basic Path Testing:

Introduction:

Path testing is a structural testing method **that involves using the source code of a program in order to find every possible executable path**. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to

reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

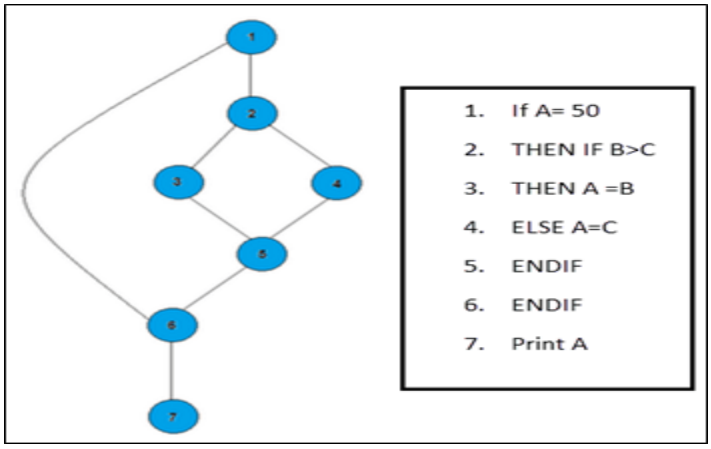
It is White-box testing technique proposed by Tom McCabe. Enables the test case designer to **derive a logical complexity of a procedural design(system).** And then use this measure **to define basis set of execution paths developer uses this**. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

Basis Path Testing in software engineering is a White Box Testing method in which test cases are defined based on flows or logical paths that can be taken through the program.

The objective of basis path testing:

is to define the number of independent paths, so the number of test cases needed can be defined explicitly to maximize test coverage.

Here we will take a simple example, to get a better idea what is basis path testing include



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that needs to be tested to get the output,

- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

Steps for Basis Path testing

The basic steps involved in basis path testing include;

- **Draw a control graph (flow graph)** (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to **determine the number of independent paths)**
- **Find a basis set of paths**
- **Generate test cases to exercise each path**

➢ Flow Graph Notation:

- ✓ A circle in a graph represents a node, which stands for a sequence of one or more procedural statements.
- ✓ A node containing a simple conditional expression is referred to as a predicate node.
- ✓ Each compound condition in a conditional expression containing one or more Boolean operators (e.g., and, or) is represented by a separate predicate node.
- ✓ A predicate node has two edges leading out from it (True and False).

- ✓ An edge, or a link, is a an arrow representing flow of control in a specific direction.
- ✓ An edge must start and terminate at a node.
- ✓ An edge does not intersect or cross over another edge.
- ✓ Areas bounded by a set of edges and nodes are called regions.
- ✓ When counting regions, include the area outside the graph as a region too.

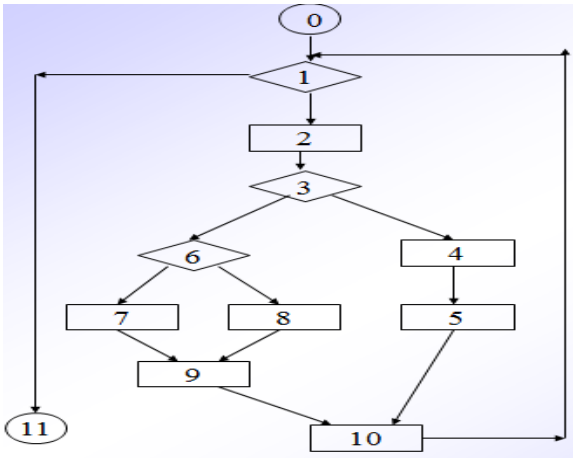


Figure 7.1: Procedural Design using Flowchart

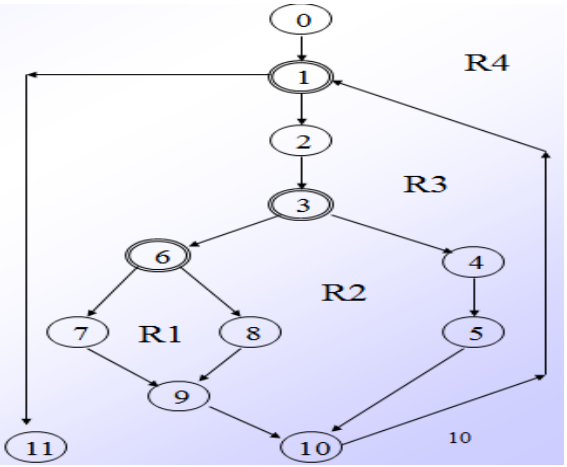


Figure 7.2: Flow Graph Notation of given Flowchart

- ➢ **Independent Program Path:**
  - ✓ Defined as a path through the program from the start node until the end node that introduces at least one new set of processing statements or a new condition (i.e., new nodes)
  - ✓ Must move along at least one edge that has not been traversed before by a previous path
  - ✓ Basis set for flow graph on previous slide
    - **Path 1: 0-1-11**
    - **Path 2: 0-1-2-3-4-5-10-1-11**
    - **Path 3: 0-1-2-3-6-8-9-10-1-11**
    - **Path 4: 0-1-2-3-6-7-9-10-1-11**

- ✓ The number of paths in the basis set is determined by the **cyclomatic complexity**

- **cyclomatic complexity Can be computed three ways**
  - ✓ **The number of regions**
  - ✓ V(G) = E – N + 2, where E is the number of edges and N is the number of nodes in graph G
  - ✓ V(G) = P + 1, where P is the number of predicate nodes in the flow graph G
  - ✓ **Number of regions**: Bounded + unbounded rejoins of flow graph

    Results in the following equations for the example flow graph
    - Number of regions = 4 (R1,R2,R3 and R4)
    - V(G) = 14 edges – 12 nodes + 2 = 4
    - V(G) = 3 predicate nodes + 1 = 4
    - **Therefore, No. of Independent path = 4 (For Given Flow graph)**
    - **Therefore, Prepare test cases that will force execution of all 4 paths**.

Advantages of Basic Path Testing

- It helps to reduce the redundant tests
- It focuses attention on program logic
- It helps facilitates analytical versus arbitrary case design
- Test cases which exercise basis set will execute every statement in a program at least once.

Example: Consider the given program that checks if a number is prime or not. For the following program:

1 Draw the Control Flow Graph
2 Calculate the Cyclomatic complexity using all the methods
3 List all the Independent Paths
4 Design test cases from independent paths

```
int main()
{
   int n, index;
   cout << "Enter a number: " << endl;
   cin >> n;
   index = 2;
   while (index <= n - 1) {
      if (n % index == 0) {
         cout << "It is not a prime number" << endl;
         break;
      }
      index++;
   }
   if (index == n)
      cout << "It is a prime number" << endl;
} // end main
```

Solution:

1. Draw the Control Flow Graph –

Step-1:

Start numbering the statements after declaration of the variables (if no variables have been initialized in that statement). However, if a variable has been initialized and declared in the same line, then numbering should start from that line itself.

For the given program, this is how numbering will be done:

```
int main()
{
 1 int n, index;
2  cout << "Enter a number: " <> n;
3   index = 2;
4   while (index <= n - 1)
```

```
5   {
6      if (n % index == 0)
7      {
8         cout << "It is not a prime number" << endl;
9         break;
10     }
11     index++;
12  }
13  if (index == n)
14     cout << "It is a prime number" << endl;
15 } // end main
```

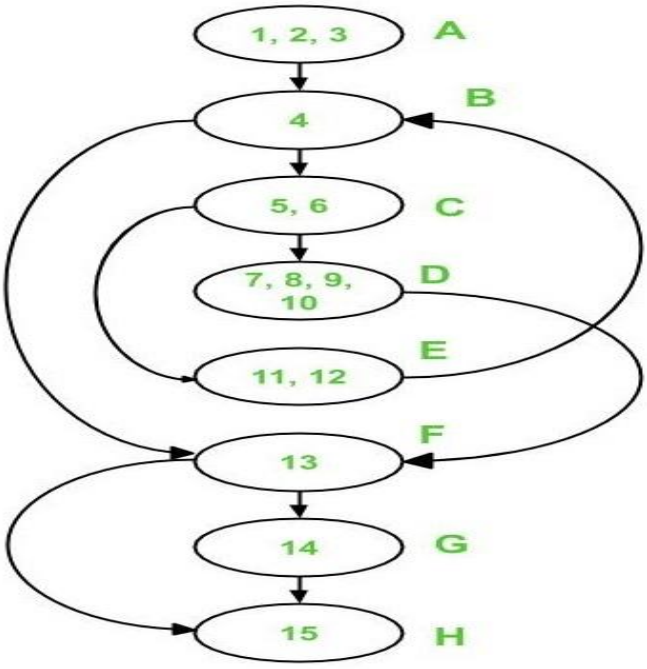
Step-2:

Put the sequential statements into one single node. For example, statements 1, 2 and 3 are all sequential statements and hence should be combined into a single node. And for other statements, we will follow the notations as discussed here.

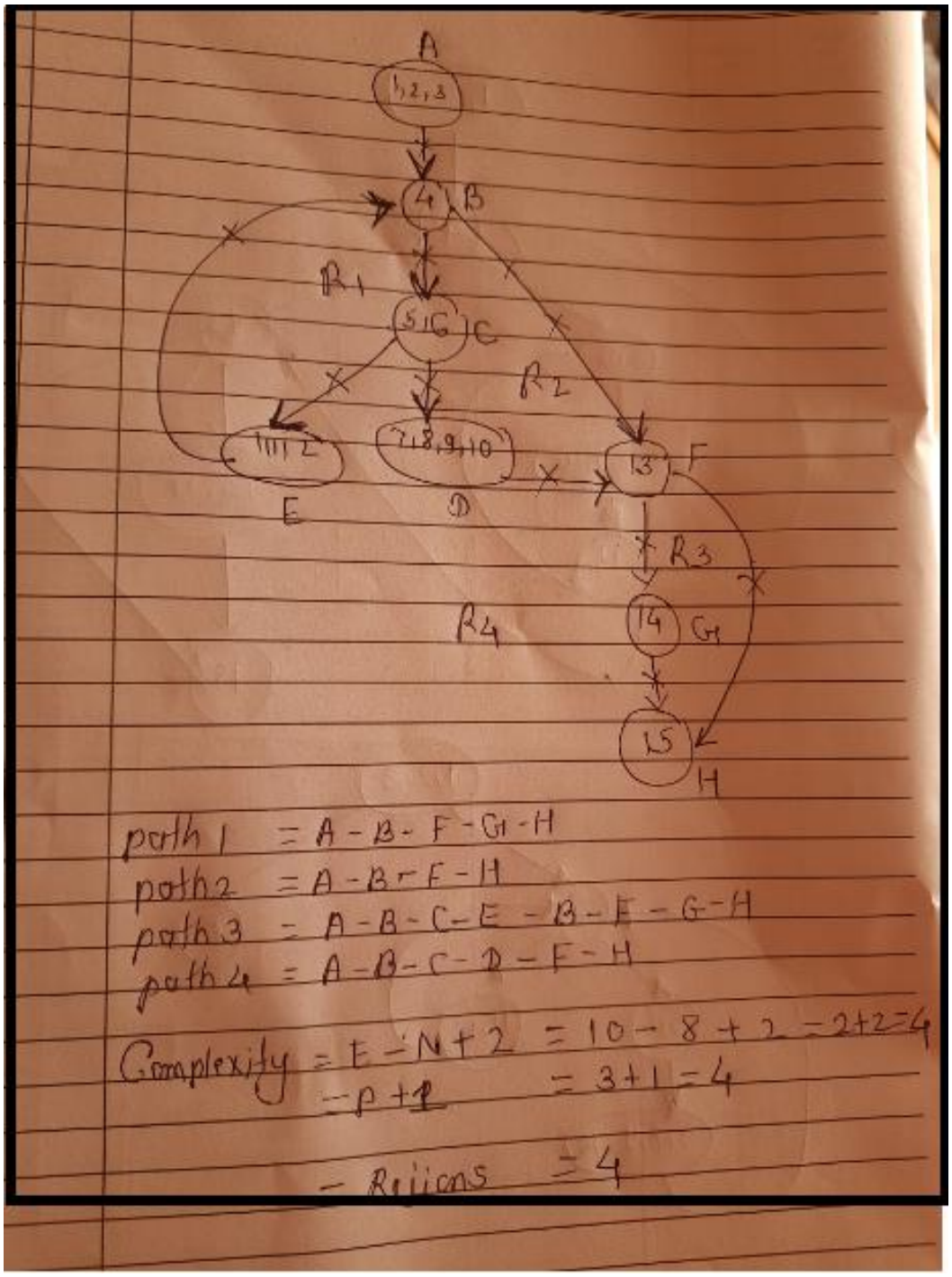
Note –

Use alphabetical numbering on nodes for simplicity.

The graph obtained will be as follows:



OR


A
1,2,3
4
B
R1
5,6
C
R2
11,12
7,8,9,10
13
F
E
D
R3
R4
14
G
15
H
path 1 = A-B-F-G-H
path 2 = A-B-F-H
path 3 = A-B-C-E-B-F-G-H
path 4 = A-B-C-D-F-H
Complexity = E-N+2 = 10-8+2 = 2+2=4
= P+1 = 3+1=4
= Regions = 4

2. Calculate the Cyclomatic complexity:

Method-1:

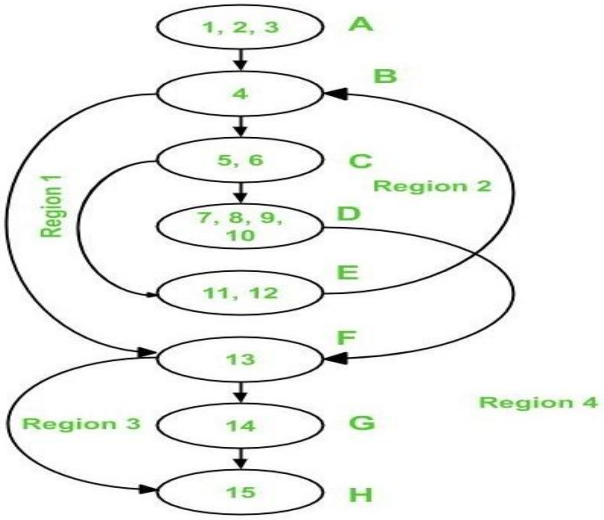
V(G) = e - n + 2
In the above control flow graph,
where, e = 10, n = 8 and p = 1
Therefore,
Cy-clomatic Complexity V(G)
= 10 - 8 + 2
= 4

Method-2:

V(G) = P+1
In the above control flow graph,
where, P = 3 (Node B, C and F)
Therefore,
Cyclomatic Complexity V(G)
= 3 + 1
= 4

Method-3:

V(G) = Number of Regions
In the above control flow graph, there are 4 regions as shown below :



Therefore, there are 4 regions: R1, R2, R3 and R4

Cyclomatic Complexity V(G)

= 1 + 1 + 1 + 1

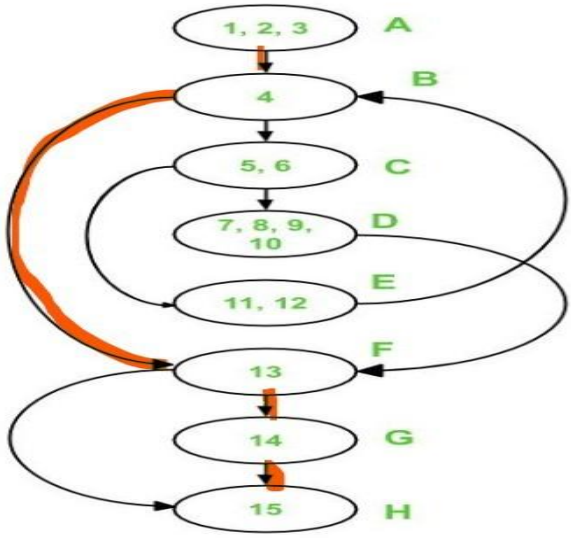
= 4

Independent Paths:

As the cyclomatic complexity V(G) for the graph has come out to be 4, therefore there are 4 independent paths.

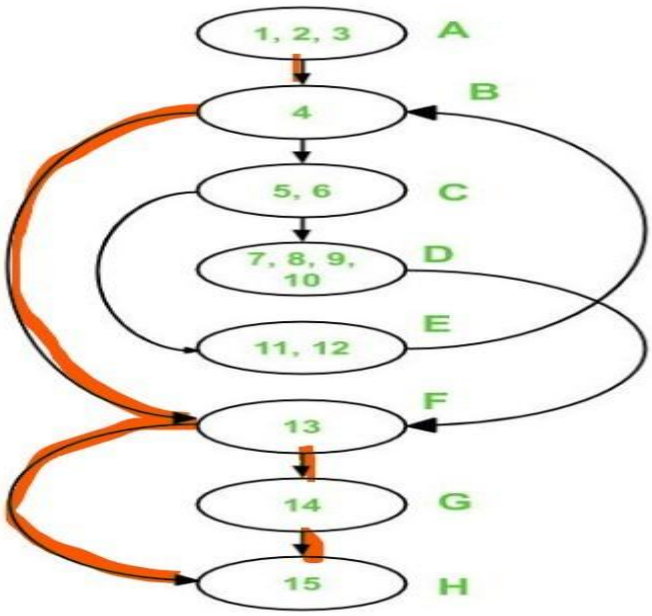
Edges covered (marked with red) by Path 1 are:

Path 1 : A - B - F - G - H



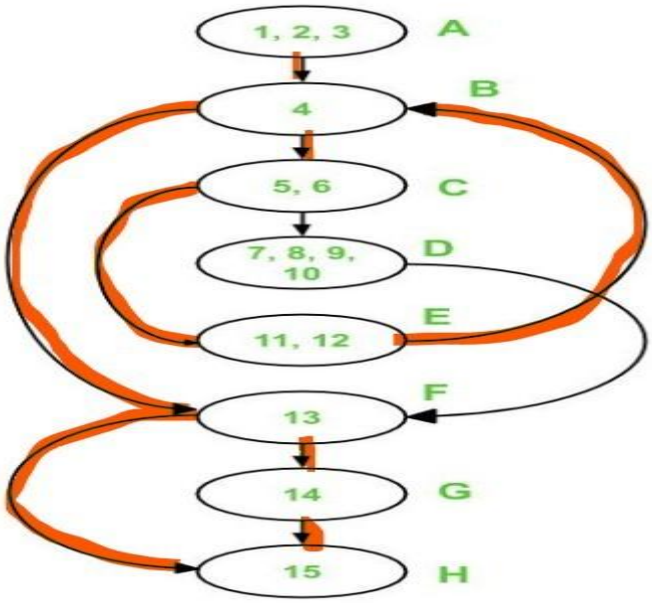
Edges covered by Path 1 and Path 2 are shown below:

Path 2 : A - B - F - H



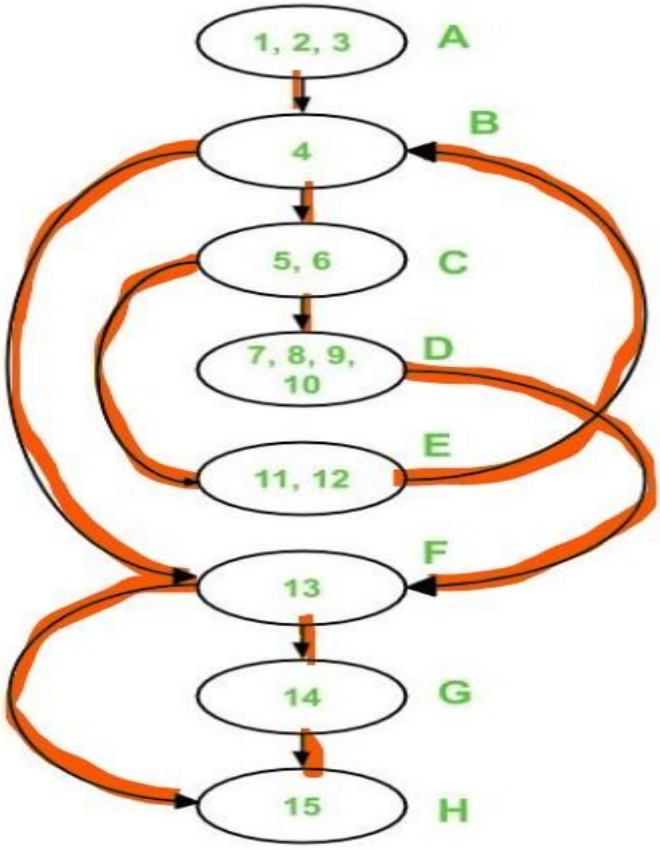
Edges covered by Path 1, Path 2 and Path 3 are :

Path 3 : A - B - C - E - B - F - G - H



Now only 2 edges are left uncovered i.e. edge C-D and edge D-F. Hence, Path 4 must include these two edges.

Path 4 : A - B - C - D - F - H



Each of these paths have introduced at least one new edge which has not been traversed before.

Note –
Independent paths are not necessarily unique.

4. Test cases:
To derive test cases, we have to use the independent paths obtained previously. To design a test case, provide input to the program such that each independent path is executed. For the given program, the following test cases will be obtained:

Test case ID	Input Number	Output	Independent path covered
Test case ID	Input Number	Output	Independent Path covered
1	1	No output	A-B-F-H
2	2	It is a prime number	A-B-F-G-H

3	3	It is a prime number	A-B-C-E-B-F-G-H
4	4	It is not a prime number	A-B-C-D-F-H

2) Control Structure Testing

Control structure testing is a group of white-box testing methods.

I. **Branch Testing (For every decision, each branch needs to be executed at least once)**
II. **Condition Testing**
III. **Data Flow Testing**
IV. **Loop Testing**

i) Branch Testing:

- **Branch Testing**:- For every decision, each branch needs to be executed at least once also called decision testing.
- Shortcoming - ignores implicit paths that result from compound conditionals.
- **Treats a compound conditional as a single statement.** (We count each branch taken out of the decision, regardless which condition lead to the branch.)
- Example: This example has two branches to be executed**:**

```
IF ( a equals b) THEN statement 1
ELSE statement 2
END IF
```

- **This example also has just two branches to be executed,** despite the compound conditional:

```
IF ( a equals b AND c less than d ) THEN statement 1
ELSE statement 2
END IF
```

- **This example has four branches to be executed**:

```
IF ( a equals b) THEN statement 1
ELSE
IF ( c equals d) THEN statement 2
ELSE statement 3
END IF
END IF
```

ii) Condition Testing: (Simple and Compound condition)

- **In condition testing all conditions in a program or module** are tested.
- **Simple Condition:**

  E1<relational operator>E2

  Where E1 and E2=Arithmetic expressions

  And Relational operators are one of the following

  <

  ≤

  =

  ≠

  ≥

  >

- **Compound Condition: It is composed of two or more simple conditions, Boolean operators and parenthesis.**

  Boolean operators included in compound conditions are

  OR-(I)

  And (&)

  Not

- **Errors in conditions can be due to**:
  - Boolean operator error

- Boolean variable error
- Boolean parenthesis error
- Relational operator error
- Arithmetic expression error

Definition: "For a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once."

Multiple-condition testing requires that all true-false combinations of simple conditions be exercised at least once.

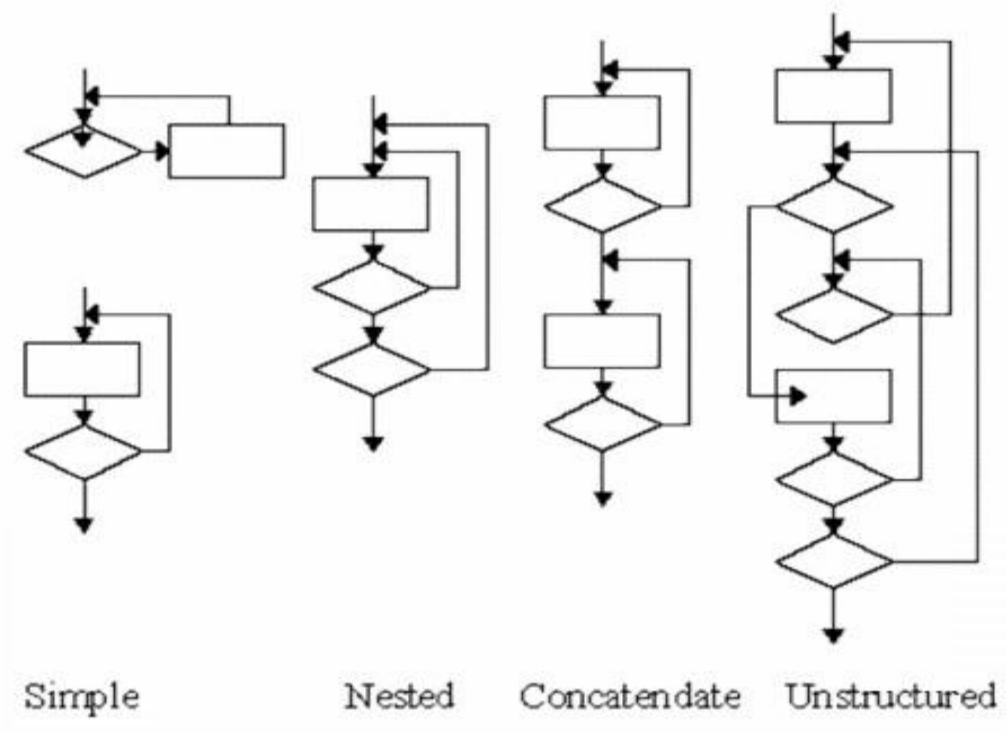
Therefore, all statements, branches, and conditions are necessarily covered.

iii) Data Flow Testing:-

Selects test paths according to the location of definitions and use of variables. This is a somewhat sophisticated technique and is not practical for extensive use. Its use should be targeted to modules with nested if and loop statements.

iv) Loop Testing:-

- Loops are fundamental to many algorithms and need testing.
- Identify different possible loops and perform testing.
- There are four different classes of loops: **simple, concatenated, nested, and unstructured.**



Examples:

Create a set of tests that force the following situations:

- ✓ **Simple Loops**: following set of tests can be applied to simple loop, where n is the maximum number of allowable passes through the loop.
  - Skip loop entirely
  - Only one pass through loop o Two passes through loop
  - m passes through loop where m<n.< p="">
  - (n-1), n, and (n+1) passes through the loop.

- ✓ **Nested Loops**:-number of possible tests increases if level of nesting increased. And it is difficult to perform huge number of tests. For this following testing approach is used.
  - Start with inner loop. Set all other loops to minimum values.
  - Conduct simple loop testing on inner loop, while holding outer loops at their minimum iteration parameter.

- Work outwards, conduct test for next loop but keeping other loop at their minimum iteration values.
- Continue until all loops tested.

Concatenated Loops:-

- If independent loops, use simple loop testing.
- If dependent, treat as nested loops.

Unstructured loops

- Whenever possible apply testing.

Black Box Testing:

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths**. Black Box Testing mainly focuses on input and output of software applications** and it is entirely based on software requirements and specifications. **It is also known as Behavioral Testing**.

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

How to do Black-Box Testing

- Initially, the **requirements and specifications of the system are examined**.
- **Tester chooses valid inputs** (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- **Tester determines expected outputs for all those inputs.**
- **Software tester constructs test cases** with the selected inputs.
- The test cases are executed.
- **Software tester compares the actual outputs with the expected outputs**.
- Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black Box Testing:

Tools used for Black box testing largely depend on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use - QTP, Selenium
- For Non-Functional Tests, you can use - LoadRunner, Jmeter

Black Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

- **Equivalence Class Testing (Equivalence Class Partitioning):**

  - ✓ **It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.**
  - ✓ **It divides input domain to different classes of data and test cases are derived for that classes of data.**
  - ✓ This test case designing techniques checks the **input and output by dividing the input into equivalent classes**. The data must be tested at least once to ensure maximum test coverage of data. It is the exhaustive form of testing, which also reduces the redundancy of inputs.
  - ✓ **For example:** Taking inputs for a test case data for the example mentioned above will have three classes from which one data will be tested.

    Valid class: 1 to 100 (any number),

    Invalid class: -1 (checking the lowest of lowest),

    Invalid class: 101(highest of highest).

- **Boundary Value Testing:**

  - ✓ Boundary value testing is **focused on the testing values at boundaries**. **Most of the errors occur at the boundary than at the center of data.** This technique determines whether a certain range of values are acceptable by the system or not. **It is very useful in reducing the number of test cases**. It is most suitable for the systems where an input is within certain ranges.

- ✓ First equivalence partitioning takes place. While defining test cases in equivalence partitioning we select some element of that class. But in BVA input from different classes which are at boundary are selected.
- ✓ It is the widely used black-box testing, which is also the basis for equivalence testing. 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.
- ✓ **For example**: Taking inputs for a test case data for an age section should accept a valid data of anything between 1-100. **According to BVP analysis**, the software will be tested against four test data as -1, 1, 100, and 101 to check the system's response using the boundary values.

Examples of Boundary value and Equivalence class partitioning:

Example 1: Equivalence and Boundary Value

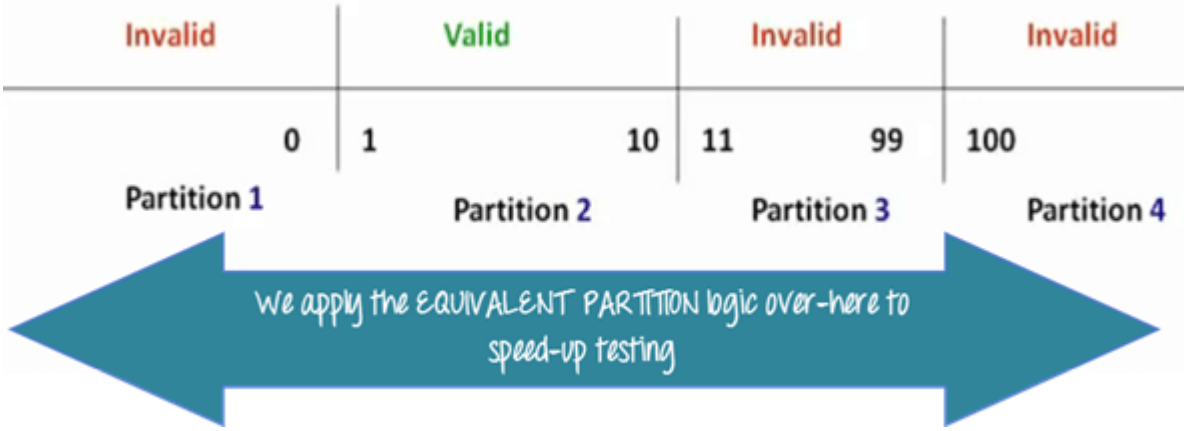
- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 are considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order Pizza: Submit

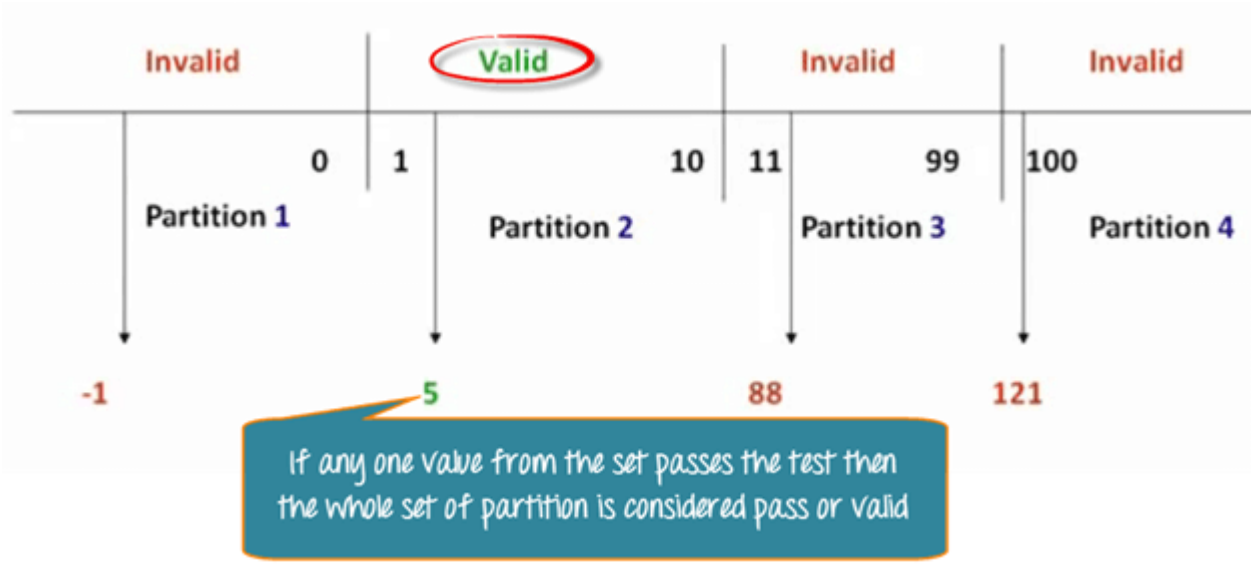
Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field (let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

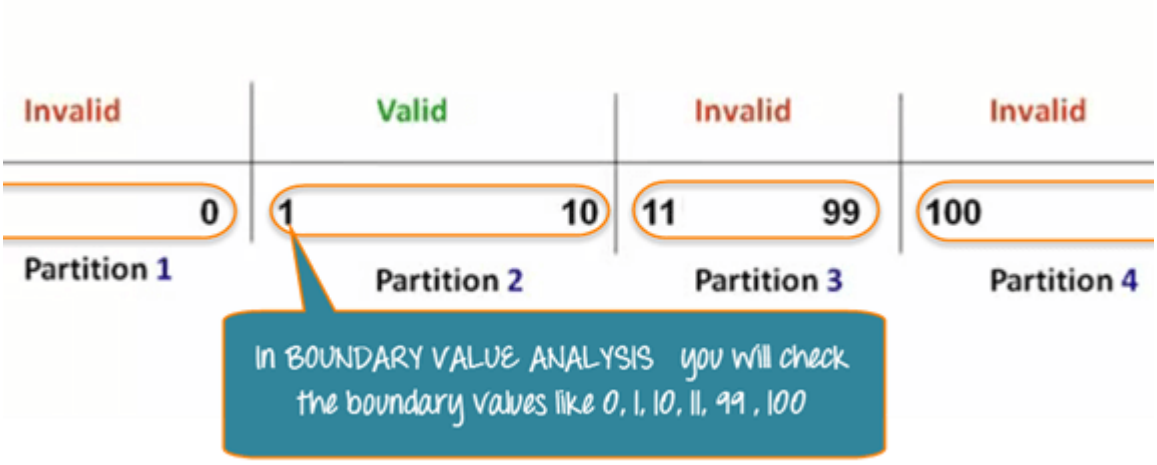
We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.



The divided sets are called Equivalence Partitions or Equivalence Classes. **Then we pick only one value from each partition for testing.** The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.



Boundary Value Analysis- in Boundary Value Analysis, you test boundaries between equivalence partitions



In our earlier equivalence partitioning example, instead of checking one value for each partition, you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at **both valid and invalid boundaries**. Boundary Value Analysis is also called **range checking**.

Equivalence partitioning and boundary value analysis (BVA) are closely related and can be used together at all levels of testing.

Example 2: Equivalence and Boundary Value

Following password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent

Enter Password: Submit

Test Scenario #	Test Scenario Description	Expected Outcome
1	Enter 0 to 5 characters in password field	System should not accept

2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Examples 3: Input Box should accept the Number 1 to 10

Here we will see the Boundary Value Test Cases

Test Scenario Description	Expected Outcome
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept
Boundary Value = 11	System should NOT accept

Why Equivalence & Boundary Analysis Testing

1. This testing is used **to reduce a very large number of test cases to manageable chunks.**
2. Very clear guidelines on determining test cases **without compromising on the effectiveness of testing.**
3. Appropriate for calculation-intensive applications with a large number of variables/inputs

Summary:

- Boundary Analysis testing is used when practically it is impossible to test a large pool of test cases individually
- In Equivalence Partitioning, first, you divide a set of test condition into a partition that can be considered.
- In Boundary Value Analysis you then test boundaries between equivalence partitions
- Appropriate for calculation-intensive applications with variables that represent physical quantities

What are the benefits of Black Box testing?

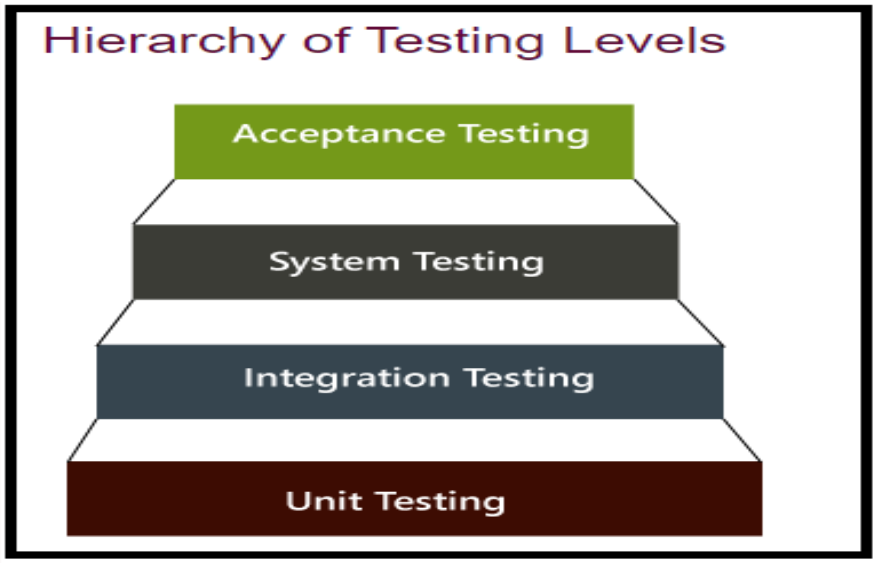
- The tester doesn't need any technical knowledge to test the system. It is essential to understand the user's perspective.
- Testing is performed after development, and both the activities are independent of each other.
- It works for a more extensive coverage which is usually missed out by testers as they fail to see the bigger picture of the software.
- Test cases can be generated before development and right after specification.
- Black box testing methodology is close to agile.

Conclusion (Black Box Testing):

- Black box testing helps to find the gaps in functionality, usability, and other features.
- This form of testing gives an overview of software performance and its output.
- It improves software quality and reduces the time to market.
- This form of testing mitigates the risk of software failures at the user's end.

6.4 Testing Levels (Strategies)

1. Unit Testing
2. Integration testing
    a. Validation Vs Verification testing
3. System Testing
    a. Usability Testing
    b. Load Testing
    c. Regression Testing
    d. Recovery Testing
    e. Migration Testing
    f. Functional Testing
    g. Non-Functional Testing
4. User Acceptance Testing
    a. Alpha testing
    b. Beta Testing



What is the Unit Test?

Unit Tests **are conducted by developers and test the unit of code** he or she developed. It is a testing method by which individual units of source code are tested to determine if they are ready to use. It helps to reduce the cost of bug fixes since the bugs are identified during the early phases of the development lifecycle.

What is Integration Test?

Integration testing **is executed by testers** and tests integration between software modules. It is a software testing technique where individual units of a program are combined and tested as a group. **Test stubs and test drivers** are used to assist in Integration Testing. Integration test is performed in two way, they are a bottom-up method and the top-down method.

OR

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple

software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

Unit Vs Integration Testing:

Unit test	Integration test
The idea behind Unit Testing is to test each part of the program and show that the individual parts are correct.	The idea behind Integration Testing is to combine modules in the application and test as a group to see that they are working fine
It is kind of White Box Testing	It is kind of Black Box Testing
It can be performed at any time	It usually carried out after Unit Testing and before System Testing
Unit Testing tests only the functionality of the units themselves and may not catch integration errors, or other system-wide issues	Integrating testing may detect errors when modules are integrated to build the overall system
It starts with the module specification	It starts with the interface specification
It pays attention to the behavior of single modules	It pays attention to integration among modules
Unit test does not verify whether your code works with external dependencies correctly.	Integration tests verify that your code works with external dependencies correctly.
It is usually executed by the developer	It is usually executed by a test team
Finding errors is easy	Finding errors is difficult
Maintenance of unit test is cheap	Maintenance of integration test is expensive

Approaches, Strategies, Methodologies of Integration Testing

Software Engineering defines variety of strategies to **execute Integration testing**, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following
  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Testing

Big Bang Testing is an Integration testing approach in which all the components or modules are integrated together **at once and then tested as a unit**. This **combined set of components** is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Testing

In the **Incremental Testing** approach, **testing is done by integrating two or more modules that are logically related to each other** and then tested for proper functioning of the application. **Then the other related modules are integrated incrementally** and the process continues until all the logically related modules are integrated and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

- **Bottom Up Integration**
- **Top Down Integration**

Stubs and Drivers

Stubs and Drivers **are the dummy programs** in Integration testing used to facilitate the software testing activity. **These programs act as substitutes for the missing models** in the testing. **They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing**.

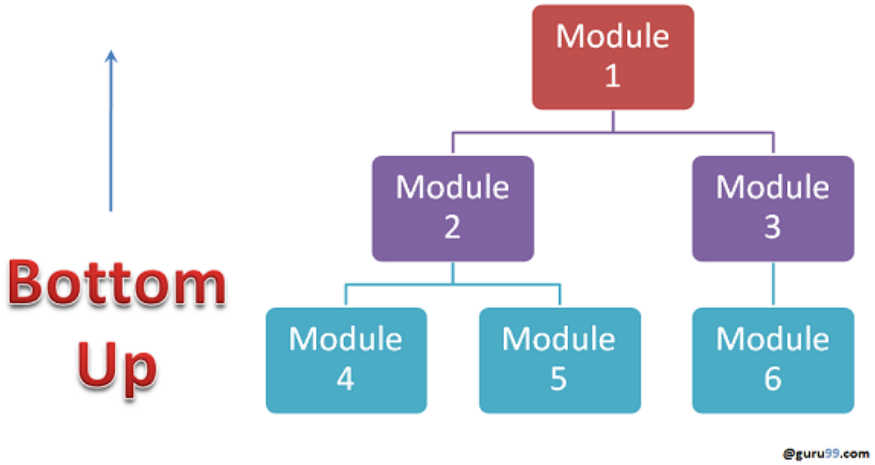
Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Bottom-up Integration Testing

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. **These tested modules are then further used to facilitate the testing of higher level modules.** The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

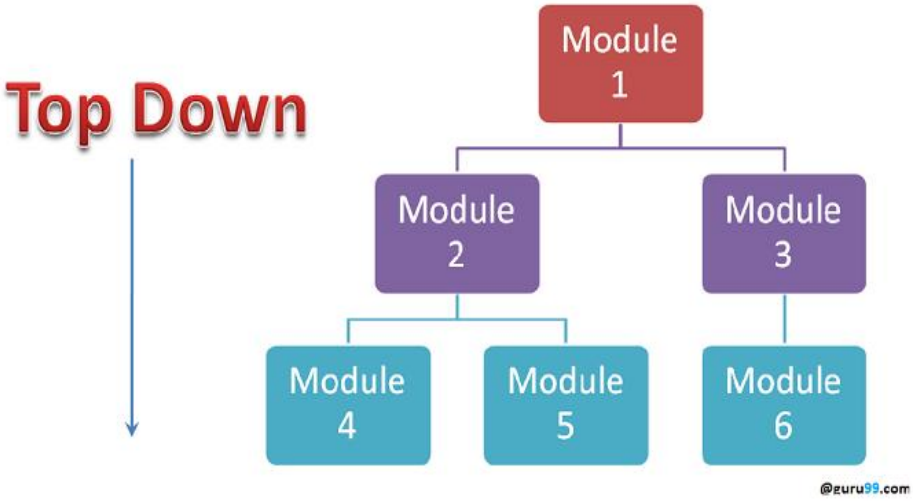
Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

Top-down incremental Testing

Top down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. **Stubs are used for testing if some modules are not ready**.

Diagrammatic Representation:



Advantages:

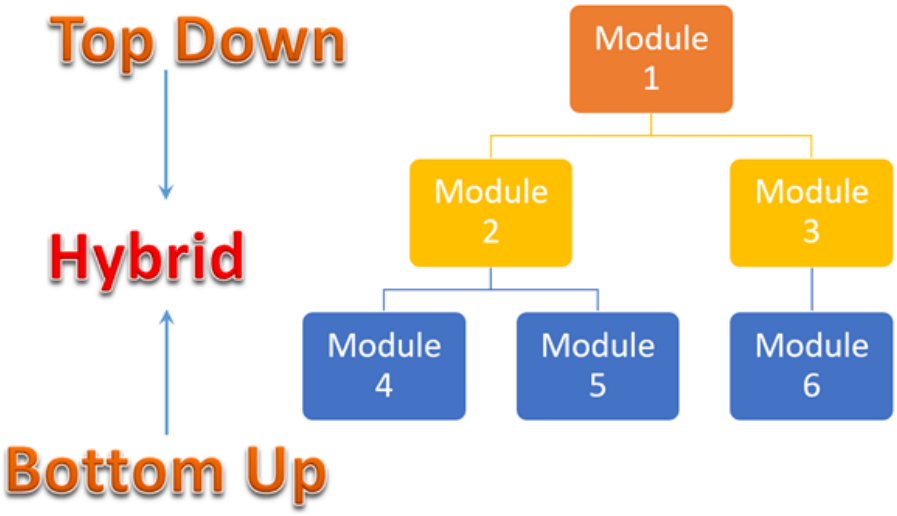
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing**. It **makes use of both stubs as well as drivers.**



How to do Integration Testing?

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

Entry and Exit Criteria of Integration Testing

Entry and Exit Criteria to Integration testing phase in any software development model

Entry Criteria:

- Unit Tested Components/Modules
- All High prioritized bugs fixed and closed
- All Modules to be code completed and integrated successfully.
- Integration tests Plan, test case, scenarios to be signed off and documented.
- Required Test Environment to be set up for Integration testing

Exit Criteria:

- Successful Testing of Integrated Application.
- Executed Test Cases are documented
- All High prioritized bugs fixed and closed
- Technical documents to be submitted followed by release Notes.

Verification in Software Testing

Verification in Software Testing is a process of checking documents, design, code, and program in order to check if the software has been built according to the requirements or not. **The main goal of verification process is to ensure quality of software application, design, architecture etc. The verification process involves activities like reviews, walk-through and inspection.**

Validation in Software Testing

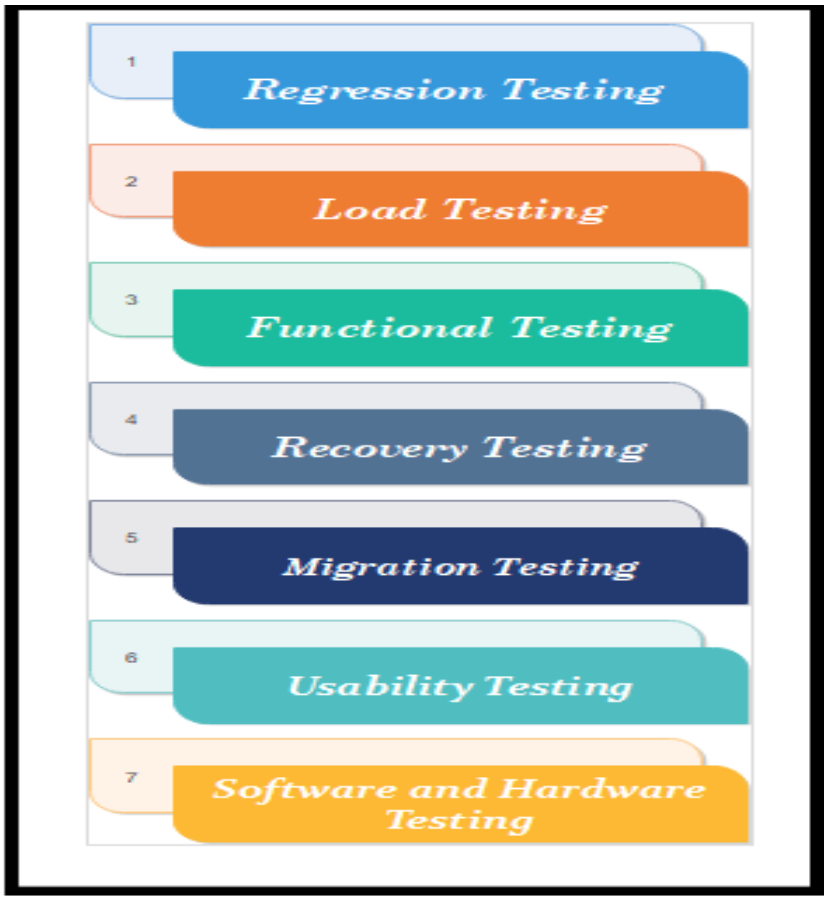
Validation in Software Testing is a dynamic mechanism of testing and validating if the software product actually meets the exact needs of the customer or not. **The process helps to ensure that the software fulfils the desired use in an appropriate environment**. **The validation process involves activities like unit testing, integration testing, system testing and user acceptance testing.**

System Testing

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. **It involves load, performance, reliability and security testing.**

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.



1. **Usability Testing**:

- ✓ Mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives.
- ✓ **Usability testing**, a non-functional **testing** technique that is a measure of how easily the system can be used by end users.
- ✓ Software Engineering, Usability Testing identifies usability errors in the system early in the development cycle and can save a product from failure.
- ✓ The goal of this testing is to satisfy users and it mainly concentrates on the following parameters of a system:

The effectiveness of the system

- Is the system is easy to learn?
- Is the system useful and adds value to the target audience?
- Are Content, Color, Icons, Images used are aesthetically pleasing?

Efficiency

- Little navigation should be required to reach the desired screen or webpage, and scrollbars should be used infrequently.
- Uniformity in the **format** of screen/pages in your application/website.
- Option to search within your software application or website.

Accuracy

- No outdated or incorrect data like contact information/address should be present.
- No broken links should be present.

User Friendliness

- Controls used should be self-explanatory and must not require training to operate
- Help should be provided for the users to understand the application/website
- Alignment with the above goals helps in effective usability testing

2. **Load Testing-**

✓ It is necessary to know that a software solution will perform under real-life loads.

✓ **Load Testing** is a non-functional software testing **process** **in which the performance of software application is tested under a specific expected load.**

- ✓ It determines how the software application behaves while being accessed by multiple users simultaneously.
- ✓ **The goal of Load Testing is to improve performance bottlenecks and to ensure stability and smooth functioning of software application before deployment.**
- ✓ **This testing usually identifies -**
  - The maximum operating capacity of an application
  - Determine whether the current infrastructure is sufficient to run the application
  - Sustainability of application with respect to peak user load
  - Number of concurrent users that an application can support, and scalability to allow more users to access it.

✓ Why Load Testing?

- Load testing gives confidence in the system & its reliability and performance.
- Load Testing helps identify the bottlenecks in the system under heavy user stress scenarios before they happen in a production environment.
- Load testing gives excellent protection against poor performance and accommodates complementary strategies for performance management and monitoring of a production environment.

✓ Goals of Load Testing:

Loading testing identifies the following problems before moving the application to market or Production:

- Response time for each transaction
- Performance of System components under various loads
- Performance of Database components under different loads

- Network delay between the client and the server
- Software design issues
- Server configuration issues like a Web server, application server, database server etc.
- Hardware limitation issues like CPU maximization, memory limitations, network bottleneck, etc.

3. **Regression Testing-**

✓ **It** involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.

✓ **REGRESSION TESTING** is defined as a type of software testing to confirm that a recent program or code change **has not adversely affected existing features**.

✓ **Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure** existing functionalities work fine.

✓ **Need of Regression Testing**

The **Need of Regression Testing** mainly arises whenever there is requirement to change the code and we need to test whether the modified code affects the other part of software application or not. Moreover, regression testing is needed, when a new feature is added to the software application and for defect fixing as well as performance issue fixing.

✓ **How to do Regression Testing**

In order **to do Regression testing** process, we need to first debug the code to identify the bugs. Once the bugs are identified, required changes are made to fix it, then the regression testing is done by selecting relevant test cases from the test suite that covers both modified and affected parts of the code.

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression Testing can be carried out using the following techniques:

✓ Selecting test cases for regression testing

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the Test Case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases -

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

6.5 Software Maintenance

- Software maintenance is the general process of changing a system after it has been delivered.
- Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.
- **Need for Maintenance –**

  Software Maintenance must be performed in order to:
  - ✓ Correct faults.
  - ✓ Improve the design.
  - ✓ Implement enhancements.
  - ✓ Interface with other systems.
  - ✓ Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.

- The change may be simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification error or accommodate new requirements.
- Software is always changing and as long as it is being used, it has to be monitored and maintained properly. This is partly to adjust for the changes within an organization but is even more important because technology keeps changing.
- Your software may need maintenance for any number of reasons – to keep it up and running, to enhance features, to rework the system for changes into the future, to move to the Cloud, or any other changes. Whatever the motivation is for software maintenance, it is vital for the success of your business. As such, software maintenance

is more than simply finding and fixing bugs. It is keeping the heart of your business up and running.

- Software maintenance is the process of changing a system after it has been delivered and is in use is called software maintenance.

  ✓ The changes may involve simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification errors.

- **Types of Maintenance:**

  There are four types of software maintenance:

  ✓ **Corrective Software Maintenance**
  ✓ **Adaptive Software Maintenance**
  ✓ **Perfective Software Maintenance**
  ✓ **Preventive Software Maintenance**

✓ **Corrective Software Maintenance**

Corrective maintenance **is concerned with fixing of reported errors in the software**. Coding errors are usually relatively cheap to correct, design errors are more expensive and requirement errors are the most expensive to repair because of extensive system redesigns.

Corrective software maintenance is what one would typically associate with the maintenance of any kind. Correct software maintenance addresses the errors and faults within software applications that could impact various parts of your software, including the design, logic, and code. **These corrections usually come from bug reports that were created by users or customers** – but corrective software maintenance can help to spot them before your customers do, which can help your brand's reputation.

✓ Adaptive Software Maintenance

Adaptive maintenance means changing software to some new environment such as different hardware platforms or for use with different operating system.

Adaptive software maintenance becomes important when the environment of your software changes. **This can be brought on by changes to the operating system, hardware, software dependencies, Cloud storage, or even changes within the operating system. Sometimes, adaptive software maintenance reflects organizational policies or rules as well.** Updating services, making modifications to vendors, or changing payment processors can all necessitate adaptive software maintenance.

✓ Perfective Software Maintenance

Perfective Software Maintenance means implementing new functional or non-functional requirement.

Perfective software maintenance focuses on the evolution of requirements and features that existing in your system. As users interact with your applications, they may notice things that you did not or suggest new features that they would like as part of the software, which could become future projects or enhancements. Perfective software maintenance takes over some of the work, both adding features that can enhance user experience and removing features that are not effective and functional. This can include features that are not used or those that do not help you to meet your end goals.

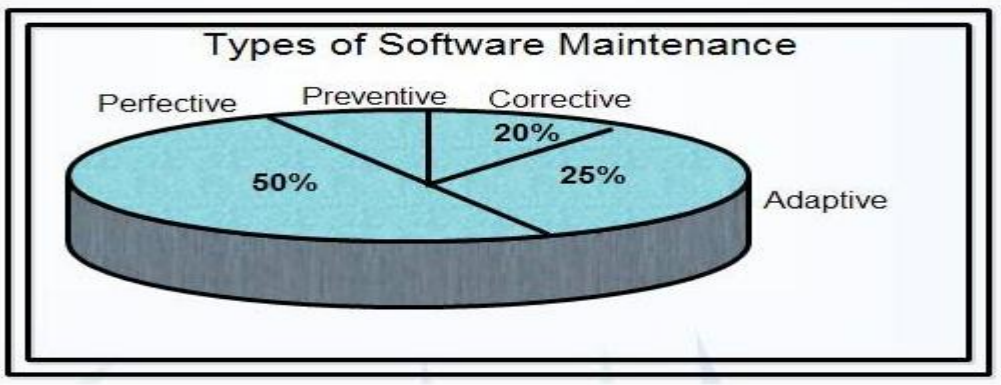
✓ Preventive Software Maintenance

Preventative Software Maintenance occurs when software is changed to improve future maintainability or reliability or to provide better basis for future enhancements.

Preventative Software Maintenance helps to make changes and adaptations to your software so that it can work for a longer period of time. The focus of the type of maintenance is to prevent the deterioration of your software as it continues to adapt and change. These services can include optimizing code and updating documentation as needed.

Preventative software maintenance helps to reduce the risk associated with operating software for a long time, helping it to become more stable, understandable, and maintainable.

- For all businesses and organizations, software maintenance is an essential part of the software development lifecycle. This isn't something that one can skip or avoid. It is absolutely necessary for the success of your software and any evolution into the future. It is important to know that maintenance needs to go much further than fixing issues or bugs – that is only one steps of the software maintenance process.
- Updating software environments, reducing deterioration, and enhancing what is already there to help satisfy the needs of all users are also included in the software maintenance examples.



6.6 Reverse Engineering

Reverse engineering, sometimes called back engineering, is a process in which software, machines, **aircraft**, architectural structures and other products **are deconstructed to extract design information from them.** Often, reverse engineering involves deconstructing individual components of larger products.

The reverse engineering process enables you **to determine how a part was designed so that you can recreate it.** Companies often use this approach when purchasing a replacement part from an original equipment manufacturer (OEM) is not an option.

The reverse engineering process is named as such because it involves working backward through the original design process. However, you often have limited knowledge about the engineering methods that went into creating the product. Therefore, the challenge is to gain a working knowledge of the original design by disassembling the product piece-by-piece or layer-by-layer.

Reverse Engineering –

Reverse Engineering is processes of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. It is also called back Engineering.

Software Reverse Engineering –

Software Reverse Engineering is the process of recovering the design and the requirements specification of a product from an analysis of it's code. **Reverse Engineering is becoming important, since several existing software products, lack**

proper documentation, are highly unstructured, or their structure has degraded through a series of maintenance efforts.

- **Why Reverse Engineering?**

✓ Providing proper system documentation.
✓ Recovery of lost information.
✓ Assisting with maintenance.
✓ Facility of software reuse.
✓ Discovering unexpected flaws or faults.

- **Used of Software Reverse Engineering –**

✓ Software Reverse Engineering is used in software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code.
✓ Reverse engineering is also useful in software testing, it helps the testers to study the virus and other malware code .