Testing and debugging

Testing:
Testing is the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by its design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

Debugging:
Debugging is the process of fixing a bug in the software. It can be defined as the identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Difference between testing and debugging

Testing	Debugging
Testing is the process to find bugs and errors.	Debugging is the process to correct the bugs found during testing.
It is the process to identify the failure of implemented code.	It is the process to give the a solution to code failure.
Testing is the display of errors.	Debugging is an error deductive process.
Testing is done by the tester.	Debugging is done by either programmer or developer.
There is no need of design knowledge in the testing process.	Debugging can't be done without proper design knowledge.
Testing can be done by insider as well as outsider.	Debugging is done only by insider. Outsider can't do debugging.
Testing can be manual or automated.	Debugging is always manual. Debugging can't be automated.
It is based on different testing levels i.e. unit testing, integration testing, system testing etc.	Debugging is based on different types of bugs.
Testing is a stage of software development life cycle (SDLC).	Debugging is not an aspect of software development life cycle; it occurs as a consequence of testing.
Testing is composed of validation and verification of software.	While debugging process seeks to match symptom with cause, by that it leads to the error correction.
Testing is initiated after the code is	Debugging commences with the execution of

Testing	Debugging
written.	a test case.

Testing strategies

Following figure shows the testing strategy:

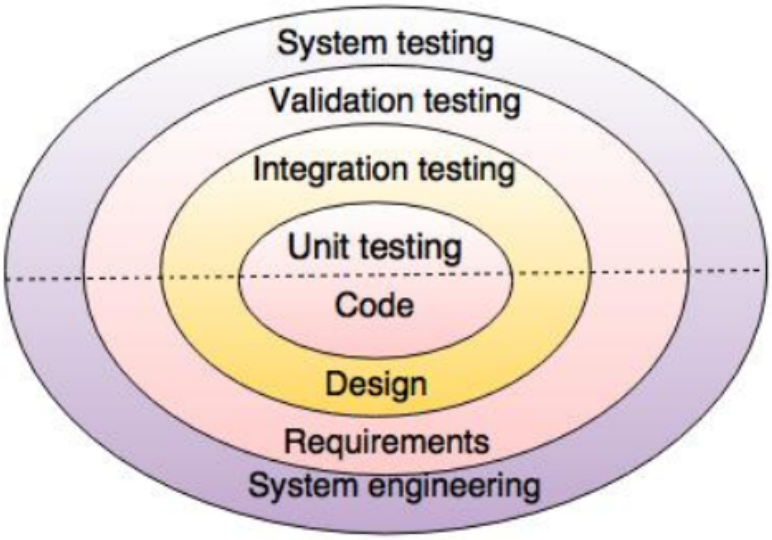


Fig. - Testing Strategy

Unit testing
Unit testing starts at the center and each unit is implemented in source code. While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration testing

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration test approaches –
There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Approach

All the modules of the system are simply put together and tested.If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.

2. Bottom-Up Integration Testing

In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested.

3. Top-Down Integration Testing

In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

4. Mixed Integration Testing –

A mixed integration testing is also called **sandwiched integration testing**. A mixed integration testing follows a combination of top down and bottom-up testing approaches. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches.

Validation testing

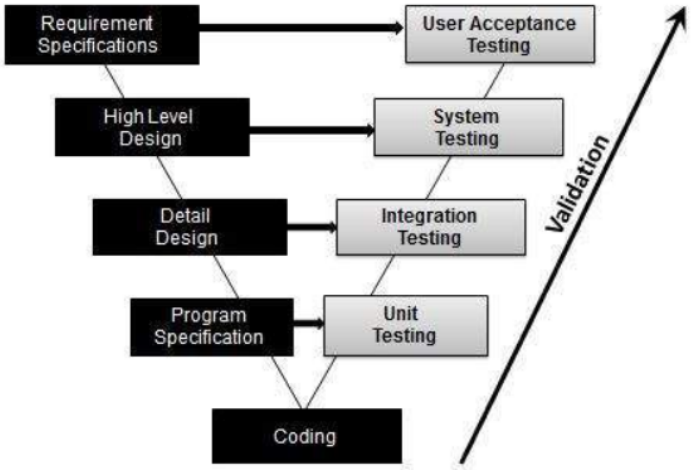
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Check all the requirements like functional, behavioral and performance requirements are validated against the construction software.

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



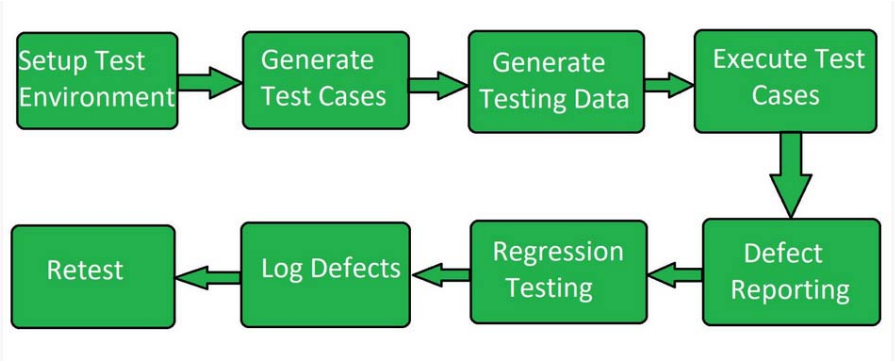
System testing

System testing confirms all system elements and performance are tested entirely. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. System testing is a black-box testing.

System Testing Process:

System Testing is performed in the following steps:

- **Test Environment Setup:**
  Create testing environment for the better-quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test to make sure none of the changes made over the course of the development process have caused new bugs.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful then again test is performed.



Different types of system testing

- **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
- **Load Testing**- is necessary to know that a software solution will perform under real-life loads.

- **Regression Testing**- 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.
- **Recovery testing** - is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup (regain) from possible crashes.
- **Migration testing-** is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
- **Functional Testing** - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions.
- **Performance Testing:**
  Performance Testing is a type of software testing that is carried out to **test the speed, scalability, stability and reliability of the software product or application.**
- **Scalability Testing:**
  Scalability Testing is a type of software testing which is carried out to check the performance of a software application or **system in terms of its capability to scale up or scale down the number of user request load.**
- **Stress Testing:**
  Stress Testing is a type of software testing performed **to check the robustness of the system under the varying loads.**

White box testing

- White box testing techniques analyze the internal structures; the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. In white box testing, code is visible to testers so it is also **called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing**. The testing can be done at system, integration and unit levels of software development.

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

How do you perform White Box Testing?

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as well.

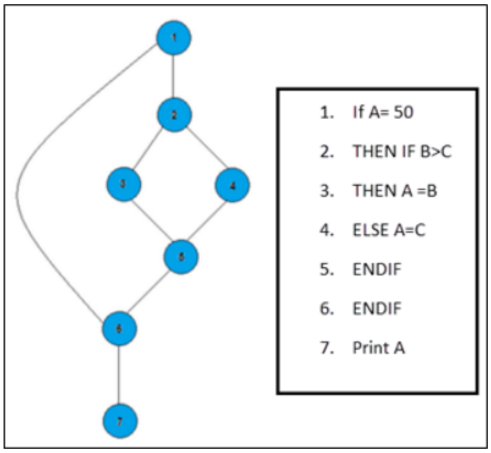
White Box Testing Techniques:

- **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
- **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.
- **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch is covered.

Basis Path Testing and Cyclomatic complexity

Basis Path Testing (Basis path testing, a structured 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. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition, Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module.

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 need to be tested to get the output,

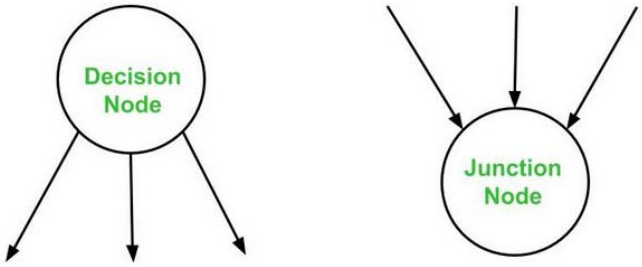
- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

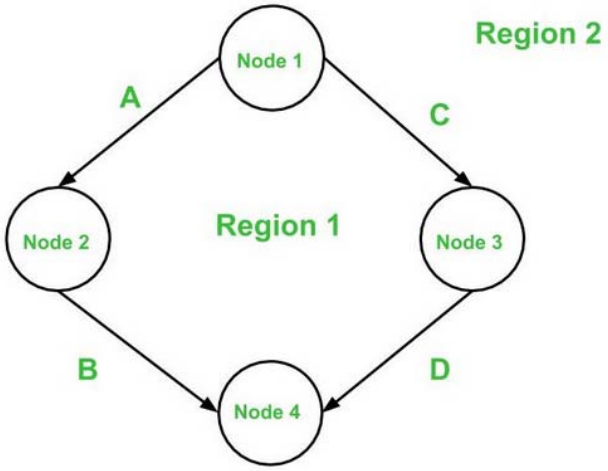
Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed:

1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

1. **Control Flow Graph –**
   A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have:

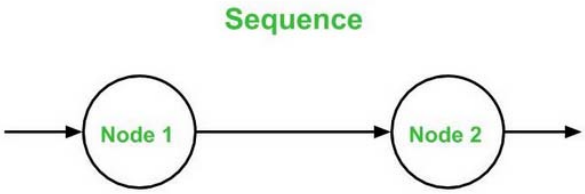
- **Junction Node –** a node with more than one arrow entering it.
- **Decision Node –** a node with more than one arrow leaving it.
- **Region –** area bounded by edges and nodes (area outside the graph is also counted as a region).



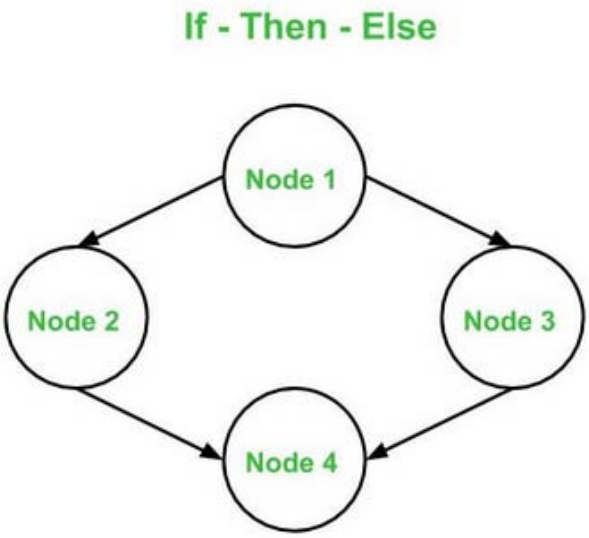


Below are the **notations** used while constructing a flow graph:

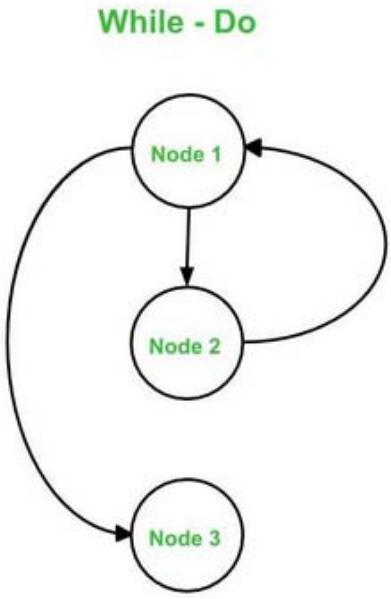
- **Sequential Statements**



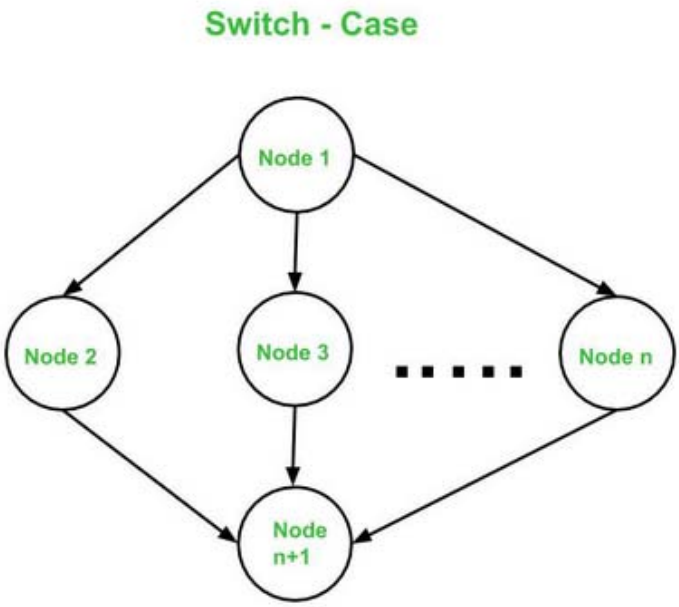
- **If – Then – Else**



- **While – Do**



- **Switch – Case**



Cyclomatic Complexity: It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity. Calculating V(G):

Formula based on edges and nodes:

V(G) = e - n + 2P
Where,
e is number of edges,
n is number of vertices,
P is number of connected components.

Formula based on Decision Nodes:
V(G) = d + P
where,
d is number of decision nodes,
P is number of connected components.

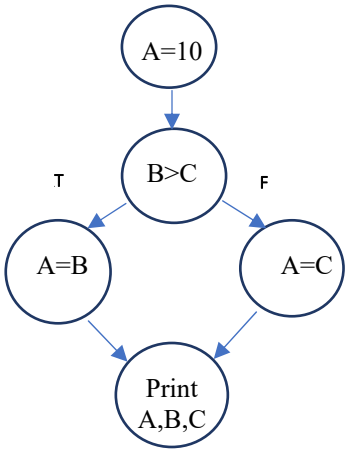
Formula based on Regions:
V(G) = number of regions in the graph

Let's take an example of section of code:

```
A = 10
IF B > C THEN
A = B
ELSE
A = C
```

ENDIF
Print A
Print B
Print C

Control Flow Graph of above code



V(G) based on edges and node
V(G) = E-N+2P = 5-5+2*1 = 2

V(G) based on decision node
V(G) = D+P = 1+1 = 2

V(G) based on region
V(G) = 2

Once the basic set is formed, TEST CASES should be written to execute all the paths.

Cyclomatic Complexity can prove to be very helpful in

- Helps developers and testers to determine independent path executions
- Developers can assure that all the paths have been tested atleast once
- Helps us to focus more on the uncovered paths
- Improve code coverage in Software Engineering
- Evaluate the risk associated with the application or program

Control structure testing

Control structure testing is used to increase the coverage area by testing various control structures present in the program. The different types of testing performed under control structure testing are as follows-

1. Branch Testing.
2. Condition Testing
3. Data Flow Testing
4. Loop Testing

5.Branch Testing

- Also called Decision Testing
- Definition: "For every decision, each branch needs to be executed at least once."
- Shortcoming - ignores implicit paths that result from compound conditionals.
- Treats a compound conditional as a single statement.

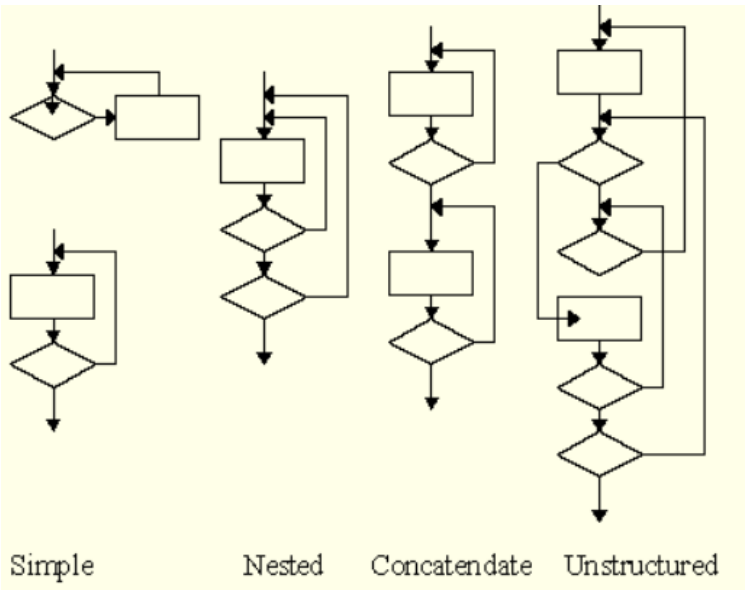
2. Condition Testing:

Condition testing is a test cased design method, which **ensures that the logical condition and decision statements are free from errors.** The errors present in logical conditions can be **incorrect boolean operators, missing parenthesis in a boolean expressions, error in relational operators, arithmetic expressions,** and so on.

3. Loop Testing:

Loops are fundamental to many algorithms and need thorough testing.

There are four different classes of loops: **simple, concatenated, nested, and unstructured.**



Create a set of tests that force the following situations:

- **Simple Loops,** where *n* is the maximum number of allowable passes through the loop.
    - Skip loop entirely
    - Only one pass through loop
    - Two passes through loop
    - m passes through loop where m<n
    - (n-1), n, and (n+1) passes through the loop
- **Nested Loops**
    - Start with inner loop. Set all other loops to minimum values.
    - Conduct simple loop testing on inner loop.
    - Work outwards
    - Continue until all loops tested.
- **Concatenated Loops**
    - If independent loops, use simple loop testing.
    - If dependent, treat as nested loops.
- **Unstructured loops**
    - Don't test - redesign.

Objectives of Loop Testing:
The objective of Loop Testing is:

- To fix the infinite loop repetition problem.
- To know the performance.
- To identify the loop initialization problems.
- To determine the uninitialized variables.

Data flow testing

Data flow testing is used **to analyze the flow of data in the program**. It is the **process of collecting information about how the variables flow the data in the program.** It tries to obtain particular information of each particular point in the process.

- The programmer can perform numerous tests on data values and variables. This type of testing is referred to as data flow testing.
- It is performed at two abstract levels: **static data flow testing and dynamic data flow testing.**
- **The static data flow** testing process involves analyzing the source code without executing it.
- Static data flow testing exposes possible defects known as data flow anomaly.
- **Dynamic data flow** identifies program paths from source code.

Let us understand this with the help of an example.

```
1. read x;
2. If(x>0)                    (1, (2, t), x), (1, (2, f), x)
3. a= x+1                     (1, 3, x)
4. if (x<=0) {                (1, (4, t), x), (1, (4, f), x)
5. if (x<1)                   (1, (5, t), x), (1, (5, f), x)
6. x=x+1; (go to 5)           (1, 6, x)
else
7. a=x+1                      (1, 7, x)
8. print a;                   (6,(5, f)x), (6,(5,t)x)
                              (6, 6, x)
                              (3, 8, a), (7, 8, a).
```

There are 8 statements in this code. In this code we cannot cover all 8 statements in a single path as if 2 is valid then **4, 5, 6, 7** are not traversed, and if 4 is valid then statement 2 and 3 will not be traversed.

Hence, we will consider two paths so that we can cover all the statements.

x= 1

Path – 1, 2, 3, 8

Output = 2

If we consider **x = 1**, in step 1; x is assigned a value of 1 then we move to step 2 (since, x>0 we will move to statement **3 (a= x+1)** and at end, it will go to statement 8 and print x =2.

For the second path, we assign x as -1

Set x= -1

Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

Output = 2

x is set as -1 then it goes to step 1 to assign x as -1 and then moves to step 2 which is false as x is smaller than 0 (x>0 and here x=-1). It will then move to step 3 and then jump to step 4; as 4 is true (x<=0 and their x is less than 0) it will jump on 5 (x<1) which is true and it will move to step 6 **(x=x+1)** and here x is increased by 1.

So,

x=-1+1

x=0

x become 0 and it goes to step 5(x<1), as it is true it will jump to step

6 (x=x+1)

x=x+1

x= 0+1

x=1

x is now 1 and jump to step 5 (x<1) and now the condition is false and it will jump to step 7 **(a=x+1)** and set a=2 as x is 1. At the end the value of a is 2. And on step 8 we get the output as 2.

Advantages of Data Flow Testing:
Data Flow Testing is used to find the following issues-

- To find a variable that is used but never defined,
- To find a variable that is defined but never used,
- To find a variable that is defined multiple times before it is used,
- Deallocating a variable before it is used.

Disadvantages of Data Flow Testing

- Time consuming and costly process
- Requires knowledge of programming languages

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 (System under test) 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.

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(Test coverage is defined as a metric in Software Testing that measures the amount of testing performed by a set of test).

  For example: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.

  Consider a software application, which **takes not less than two-digit number and not more than 3-digit number**, for its execution.

  Inputs to test are:1,2,3,.....1498,1499,1500.

  4 classes are formed and accordingly the inputs are divided into category of valid and invalid inputs.

  - Any number which is not less than two-digit number and not more than 3-digit number is valid.
  - Any number which is less than two-digit number is invalid.
  - Any number which more than two-digit number is invalid.
  - Every other character which is other than number is invalid.

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

- **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. 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.
- **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column. Decision Table Technique is a systematic approach where **various input combinations and their respective system behavior arecaptured in a tabular form**. It is appropriate for the functions that have a logical relationship between two and more than two inputs.

Example: How to make Decision Base Table for Login Screen

Let's create a decision table for a login screen.

The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend:

- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Interpretation:

- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

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.

Black Box Testing	White Box Testing
• It is functional test of the software.	• It is structural test of the software.
• 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.
• It is applicable to the higher levels of testing of software.	• It is generally applicable to the lower levels of software 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.

Types of Black Box Testing:

- A. Functional Testing
- B. Non-functional testing
- C. Regression Testing

Types of White Box Testing:

- A. Path Testing
- B. Loop Testing
- C. Condition testing