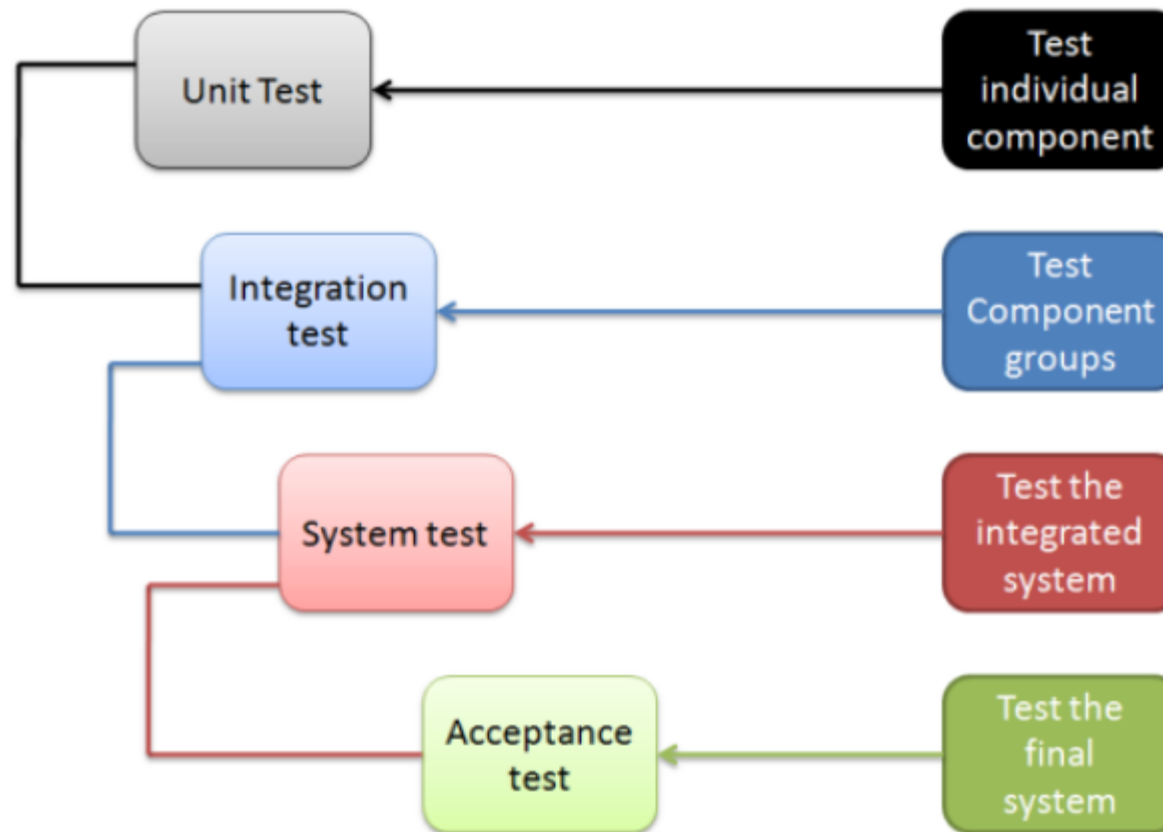


Different levels of software testing



Black-box testing

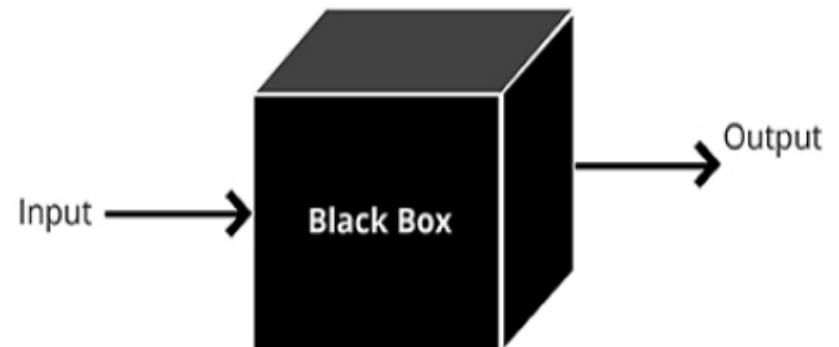
- Also known as Functional Testing.
- In Black Box Testing method testing is done without knowing the internal codes and structure of the program.
- Tester knows only about the inputs and the expected outputs of the application.

Blackbox Testing Techniques

There are different techniques involved in Black Box testing.

- ❖ Equivalence Class
- ❖ Boundary Value Analysis
- ❖ Domain Tests
- ❖ Orthogonal Arrays
- ❖ Decision Tables
- ❖ State Models
- ❖ Exploratory Testing
- ❖ All-pairs testing

BLACK BOX TESTING APPROACH



Advantages

- It is well suited and efficient for large scale segments.
- A tester can be non-technical. Highly qualified testers are not required and hence it is less expensive.
- Tests are done from user's point of view and are useful to verify the differences in the actual and system and specifications.
- Tests can be defined as soon as specifications are complete.

Disadvantages

- The coverage is limited since only a selected number of tests are actually performed.
- As the tester has limited knowledge of the software, the testing is inefficient.
- The coverage is blind as the tester cannot target specific codes.

White-box testing

- White Box Testing is also known as Glass Box Testing, Clear Box Testing, Open Box Testing, Logic Driven Testing, Path Driven Testing or Structural Testing.
- It is the testing method in which internal codes & structure of the software is known to the tester.
- The main focus here is on strengthening the security and on improving design and usability of the software.

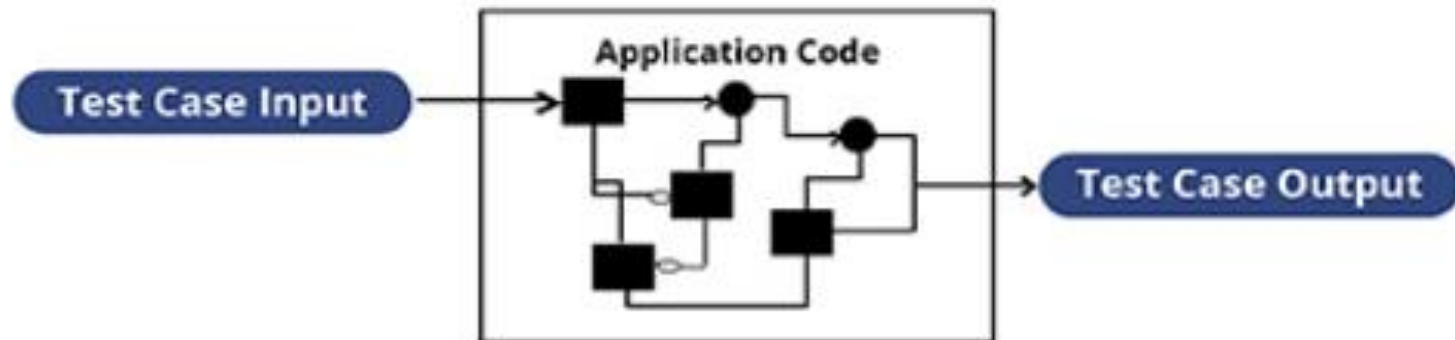
Steps in White-box testing

- Understanding the source code
- Create tests and execute

White Box Testing Techniques

- Statement Coverage
- Branch Coverage
- Path Coverage

WHITE BOX TESTING APPROACH



Advantages

- Testing is more thorough, as it covers all possible paths of code.
- As the tester is aware of internal coding structure, it is helpful to decide which type of input data is needed for testing software application effectively.
- White Box Testing allows you to help in the code optimization.

Disadvantages

- As a highly skilled resource is required to carry out testing who know the deep knowledge of internal structure of the code, the cost is high.
- If the application under test is large in size, then exhaustive testing is impossible.
- It is not possible for testing to test each and every path/condition of software program, which might miss the defects in code.



Black Box
Testing

V/s



White Box
Testing

Equivalence Class Testing

Equivalence Class Testing is one of the black-box testing design technique.

The input domain of program is partitioned into finite number of classes.

The equivalence classes are identified by taking each input condition and partitioning into valid and invalid classes.

Ex: $1 < x < 999$

Equivalence Class Testing

The input and the output domain is partitioned into mutually exclusive parts called Equivalence classes.

Any one sample from a class is representative of entire class.

Invalid	Valid	Invalid	Invalid		
0	1	10	11	99	100
Partition 1	Partition 2	Partition 3	Partition 4		

Steps to Make Test Cases

- Identify Equivalence Classes based on input condition and output conditions and partition it into valid and invalid classes.
- Generate test cases using Equivalence classes.

Example 1

- Assume, we have to test a field which accepts Age 18 – 56

AGE

Enter Age

*Accepts value 18 to 56

EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
≤ 17	18-56	≥ 57

Example 1 Analysis

- Valid Input: 18 – 56
- Invalid Input: less than or equal to 17 (≤ 17), greater than or equal to 57 (≥ 57)
- Valid Class: 18 – 56 = Pick any one input test data from 18 – 56
- Invalid Class 1: ≤ 17 = Pick any one input test data less than or equal to 17
- Invalid Class 2: ≥ 57 = Pick any one input test data greater than or equal to 57
- We have one valid and two invalid conditions here.

Example 2

- Assume, we have to test a field which accepts a Mobile Number of ten digits.

MOBILE NUMBER

Enter Mobile No.

*Must be 10 digits

EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
987654321	9876543210	98765432109

Example 2 Analysis

- Valid input: 10 digits
- Invalid input: 9 digits, 11 digits
- Valid Class: Enter 10 digit mobile number = 9876543210
- Invalid Class Enter mobile number which has less than 10 digits = 987654321
- Invalid Class Enter mobile number which has more than 11 digits = 98765432109

Uses of Equivalence Class testing

Equivalence Partitioning refers to a testing technique, with two prime goals like:

- 1) Reduction of number of test cases to the bare minimum.
- 2) Identification of the ideal test cases, which should be able to cover maximum number of scenarios.

Other uses include,

Coverage of all possible scenarios.

Its possible to do Equivalence class testing at every test-level.

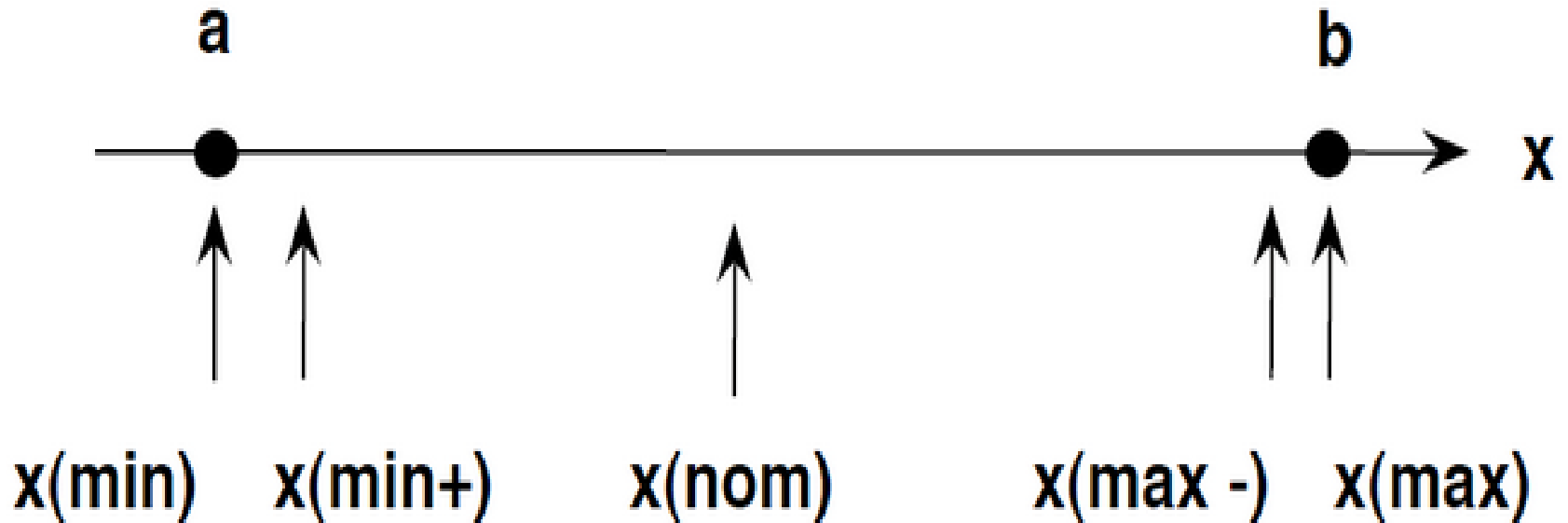
Boundary Value Testing

- Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.
- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:
 - Minimum
 - Just above the minimum
 - A nominal value
 - Just below the maximum
 - Maximum

Need for BVA

- Programmer forgets to use `<=` instead uses `<`
- Programmer forgets to count 0 while using a counter variable
- Mostly used for testing applications written in strongly-typed languages.

Boundary Value Testing



Example –

If we want to test a field which should accept only amount more than 10 and less than 20

Example 1

We test a module developed for HR, which determines to hire the candidate or not, on the age of the candidate:

0-16 – Do not hire

16-18 – Can hire only part-time

18-55 – Can hire on full-time

55-99 – Do not hire

Equivalence classes:	Boundary Values
0-15: Do not Hire	{-1, 0, 1}, {14, 15, 16},
16-17: Can hire only part time	{17, 18, 19},
18-54: Can hire on full time	{54, 55, 56},
55-99: Do not Hire	{98, 99, 100}

Example 2

- 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:

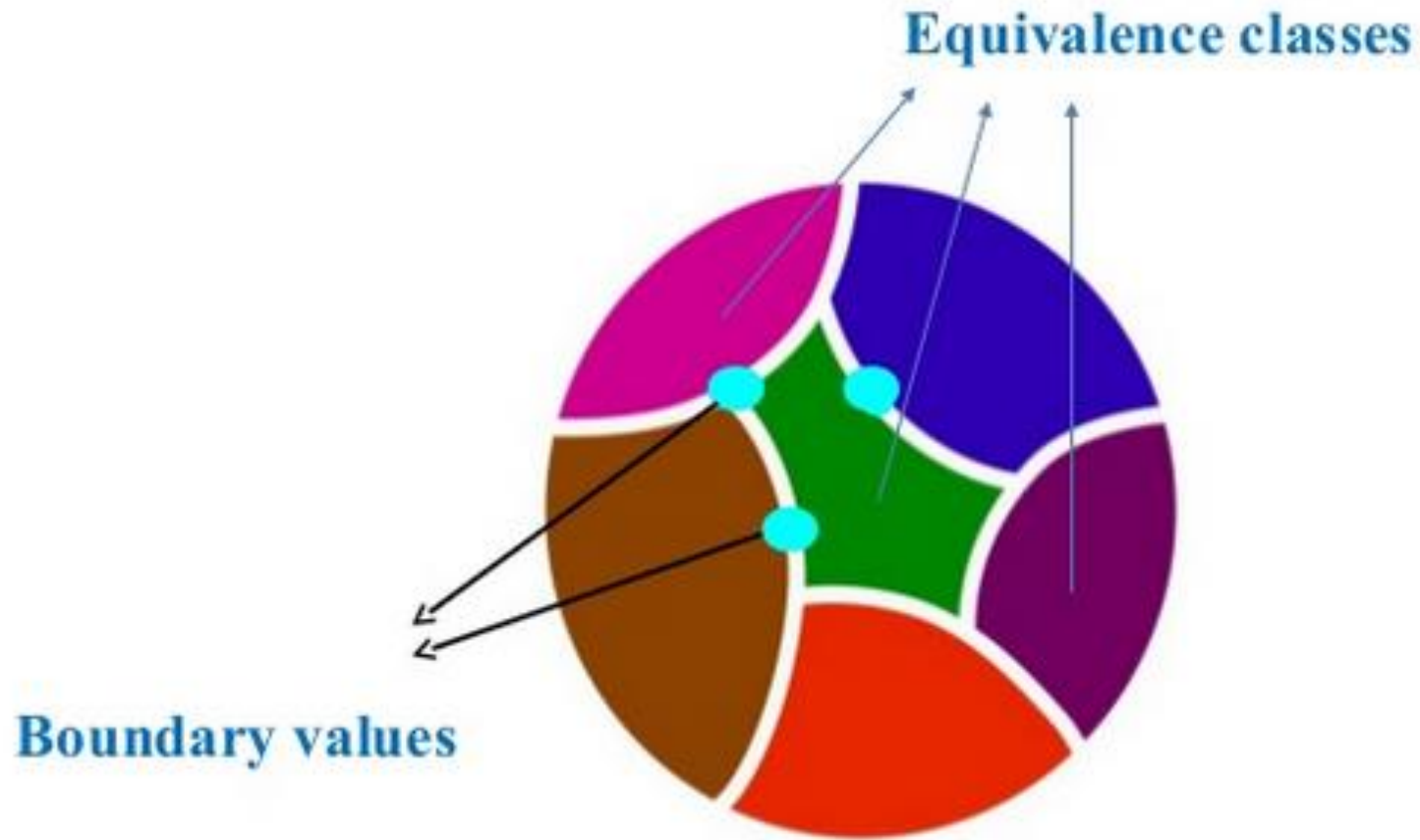
Submit Query

Test Scenario #	Test Scenario Description	Expected Outcome
-----------------	---------------------------	------------------

Uses of Boundary Value Testing

- Boundary Analysis testing is used when practically it is impossible to test a large pool of test cases individually.
- In Boundary Value Analysis you then test boundaries between equivalence partitions.
- Appropriate for calculation-intensive applications with variables that represent physical quantities

Visualization!



Decision-based testing

- Decision table testing is a software testing technique used to test system behavior for different input combinations.
- This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form.
- That is why it is also called as a **Cause-Effect table** where Cause and effects are captured for better test coverage.

- In a system where for each set of input values the system behaviour is different, boundary value and equivalent partitioning technique are not effective in ensuring good test coverage.
- In this case, decision table testing is a good option.
- **Decision table testing** is black box test design technique to determine the test scenarios.

Uses of Decision-table testing

- It also helps in better test coverage for complex business logic.
- This technique can make sure of good coverage, and the representation is simple so that it is easy to interpret and use.

Advantages of Decision-table testing

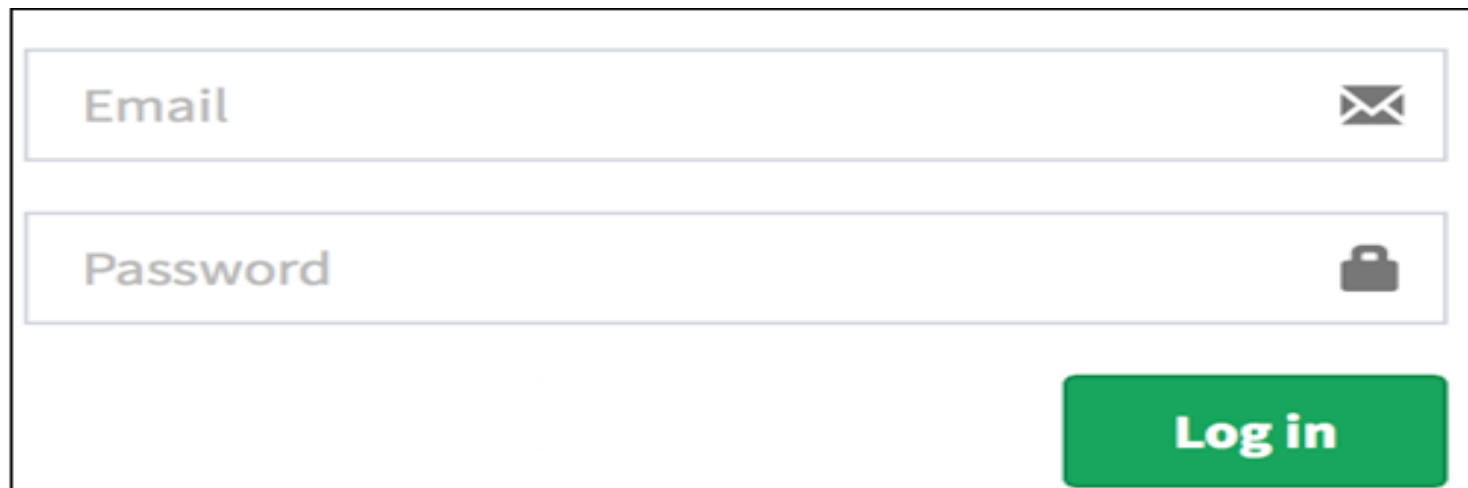
- The representation is simple so that it can be easily interpreted and is used for development and business as well.
- This table will help to make effective combinations and can ensure a better coverage for testing
- Any complex business conditions can be easily turned into decision tables
- In case we are going for 100% coverage typically when the input combinations are low, this technique can ensure the coverage.

Disadvantages of Decision-table testing

- The main disadvantage is that when the number of input increases the table will become more complex.

Example 1 - How to make Decision Base Table for 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.



A login form with two input fields and a button. The first field is labeled 'Email' and has an envelope icon on the right. The second field is labeled 'Password' and has a lock icon on the right. Below these fields is a green button with the text 'Log in'.

Field	Label	Icon
Input 1	Email	Envelope
Input 2	Password	Lock
Button	Log in	

Example 1 – Decision-table

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

Example 1 – Interpretation/Analysis

- 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

Example 2

- Let's take an example of a finance application, where users pay money – monthly Repayment or year wise (the term of loan). If user chooses both options, the system will create a negotiation between two. So, there are two conditions of the loan amount
- Recognize the exact outcome for each combination – Process loan money, Process Term

Input combinations

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	Y	Y	N	N
Terms of loan has been mentioned	Y	N	Y	N

Decision table – Combinations and outcomes

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	Y	Y	N	N
Terms of loan has been mentioned	Y	N	Y	N
Actions/Outcomes				
Process loan money	Y	Y		
Process term	Y		Y	

Data flow testing

- Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects.
- Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

The general idea – data flow testing

Data flow testing can be performed at two conceptual levels.

- 1) Static Data Flow Testing
- 2) Dynamic Data Flow Testing

Static Data Flow Testing

- Identify potential defects, commonly known as data flow anomaly.
- Analyze source code
- Do not execute code

Dynamic Data Flow Testing

- Involves actual program execution
- Identify paths to execute them.
- Paths are identified based on data flow testing criteria.

Data Flow Anomaly

- Anamoly – Abnormal way of doing something
- Ex: The second definition of x overrides the first
- $x = f1(y);$
- $x = f2(z);$
- There are three abnormal situations with using variable
- Type 1: Defined and then defined again
- Type 2: Undefined but referenced.
- Type 3: Defined but not referenced.

Dataflow testing

- Typically, Data Flow testing helps us to pinpoint any of the following issues –
- A variable that is declared but never used within the program.
- A variable that is used but never declared.
- A variable that is defined multiple times before it is used.
- Deallocating a variable before it is used.

Path testing

- 1) Path coverage tests all the paths of the program.
- 2) This is a comprehensive technique which ensures that all the paths of the program are traversed at least once.
- 3) This technique is useful for testing the complex programs, which basically involve loop statements or combination of loops and decision statements.

Path Testing

- Path testing is a Structural Testing method that involves using the source code of a program to attempt to find every possible executable path.
- Knowledge of source code is used to define the test cases and to examine outputs.

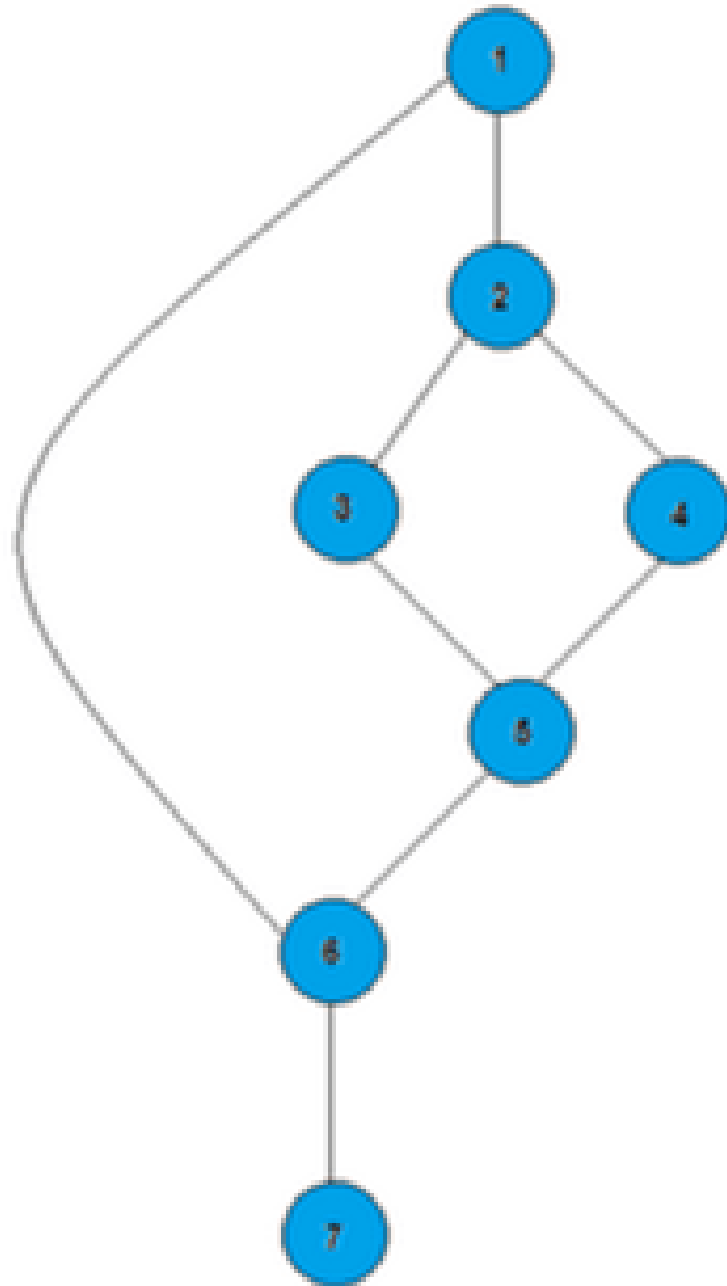
Flow graph

Given a program written in an imperative programming language, its **program graph** is a **directed graph** in which **nodes** are statement fragments, and **edges** represent flow of control.

It may be called “Control Flow Graph”

What is Path?

- A path through a program is a sequence of statements that starts at an entry, junction or decision and ends.
- A path may go through several junctions, processes or decisions, one or more times.



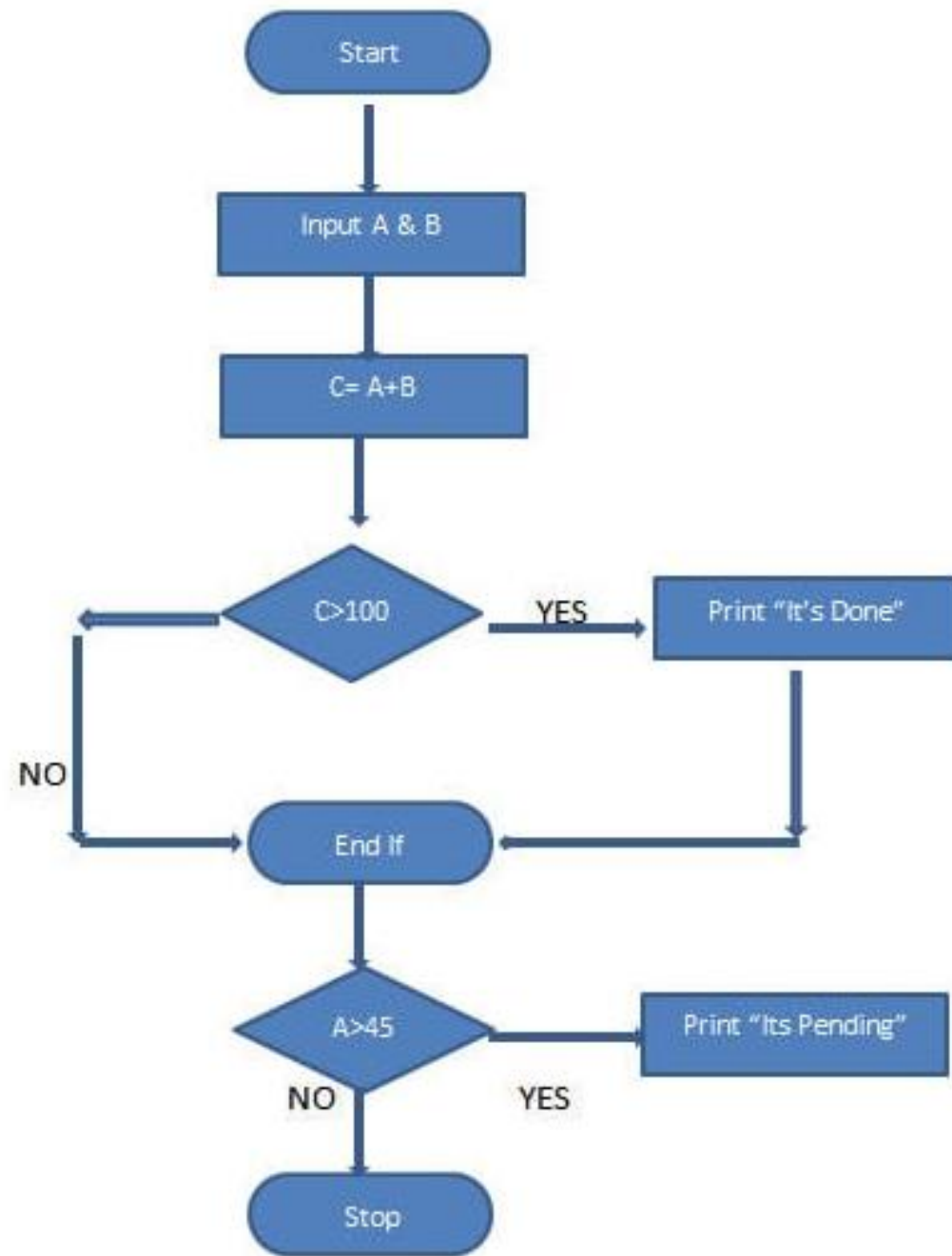
1. If A= 50
2. THEN IF B>C
3. THEN A =B
4. ELSE A=C
5. ENDIF
6. ENDIF
7. Print A

Example 1

- 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

Example 2

- Consider the below simple pseudocode
 - INPUT A AND B
 - $C = A + B$
 - IF $C > 100$
 - PRINT "ITS DONE"
 - END IF
 - IF $A > 50$
 - PRINT "IT'S PENDING"
 - END IF



Path Coverage

Testcases to be covered

- TestCase_01: A=50, B=60
- TestCase_02: A=55, B=40
- TestCase_03: A=40, B=65
- TestCase_04: A=30, B=30

Steps for Basis Path testing

- Draw a control 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

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