Software Testing Background-II

Mr .Sachin J Pukale

Lecture in Information Technology

Government Polytechnic Nagpur

Dynamic White-Box Testing

- Testing a running program and since it’s white-box, it must be about looking inside the box, examining the code, and watching it as it runs. It’s like testing the software with X-ray glasses.

- Dynamic white-box testing, in a nutshell, is using information you gain from seeing what the

1. code does and how it works to determine what to test
2. what not to test
3. and how to approach the testing.

Cont..

- Another name commonly used for dynamic white-box testing is *structural testing* because you can see and use the underlying structure of the code to design and run your tests.
- Dynamic white-box testing isn’t limited just to seeing what the code does. It also can involve directly testing and controlling the software.

Dynamic white-box testing Includes

1. Directly testing low-level functions, procedures, subroutines, or libraries.

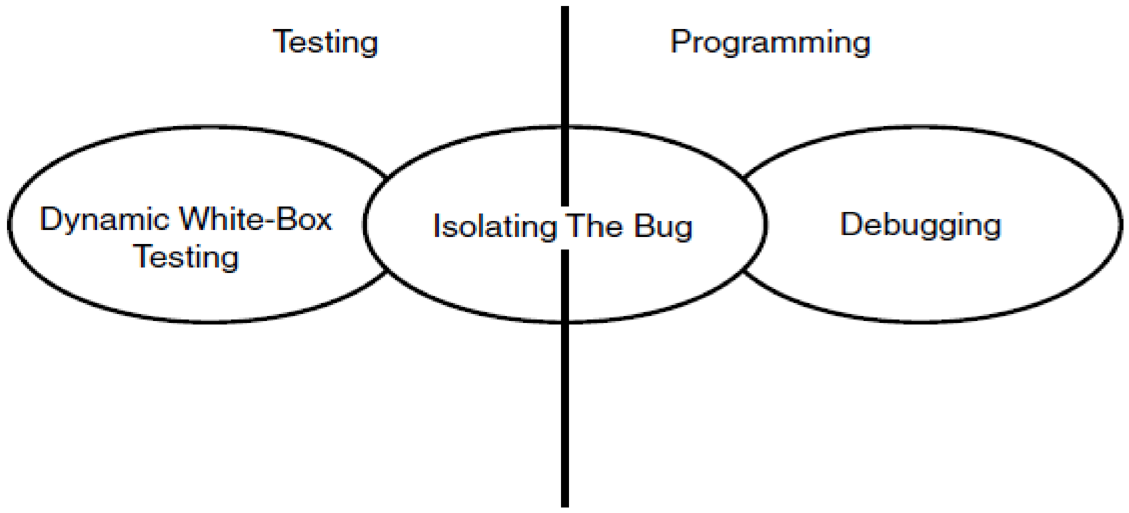
2. Testing the software at the top level, as a completed program, but adjusting your test cases based on what you know about the software's operation.

3. Gaining access to read variables and state information from the software to help you determine whether your tests are doing what you thought.

4. Measuring how much of the code and specifically what code you "hit" when you run your tests and then adjusting your tests to remove redundant test cases and add missing ones.

Dynamic White-Box Testing versus Debugging

- Dynamic white-box testing and debugging have different goals but they do overlap in the middle.
- The goal of dynamic white-box testing is to find bugs. The goal of debugging is to fix them.



PARAMETER	DEBUGGING	TESTING
Definition	Debugging is the process to correct the bugs found during testing.	Testing is the process to find bugs and errors.
Objective	It is the process to give the absolution to code failure.	It is the process to identify the failure of implemented code.
Performed by	Debugging is done by either programmer or developer.	Testing is done by the tester.
Background Knowledge	Debugging can’t be done without proper design knowledge.	There is no need of design knowledge in the testing process.
User Type	Debugging is done only by insider. Outsider can’t do debugging.	Testing can be done by insider as well as outsider.
Process Type	Debugging is always manual. Debugging can’t be automated.	Testing can be manual or automated.
Levels	Debugging is based on different types of bugs.	It is based on different testing levels i.e. unit testing, integration testing, system testing etc.
Phases of SDLC	Debugging is not an aspect of software development life cycle, it occurs as a consequence of testing.	Testing is a stage of software development life cycle (SDLC).
Perform	Debugging commences with the execution of a test case.	Testing is initiated after the code is written.

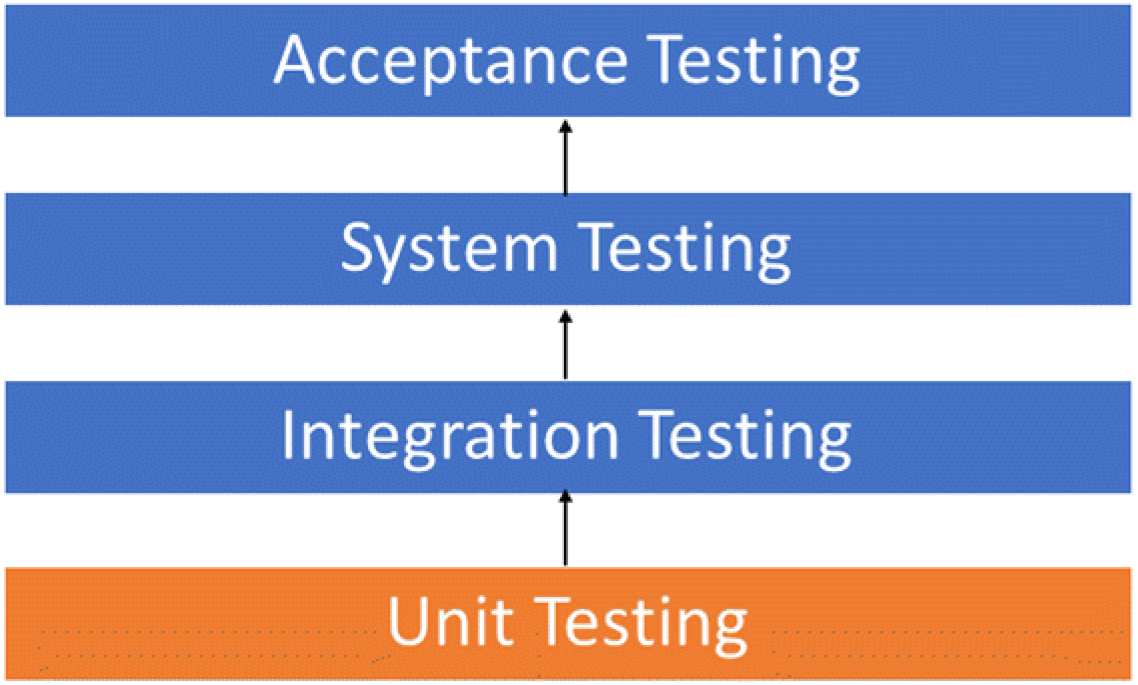
Testing the Pieces

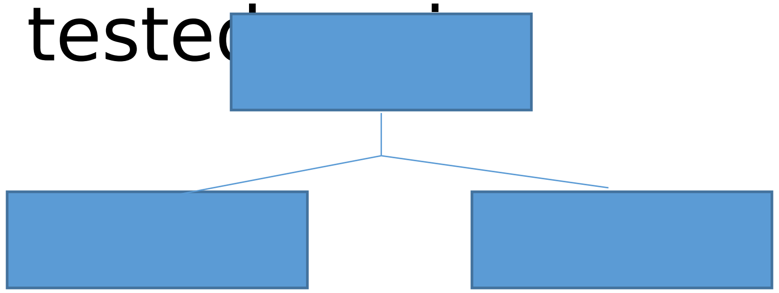
- There are two reasons for the high cost:

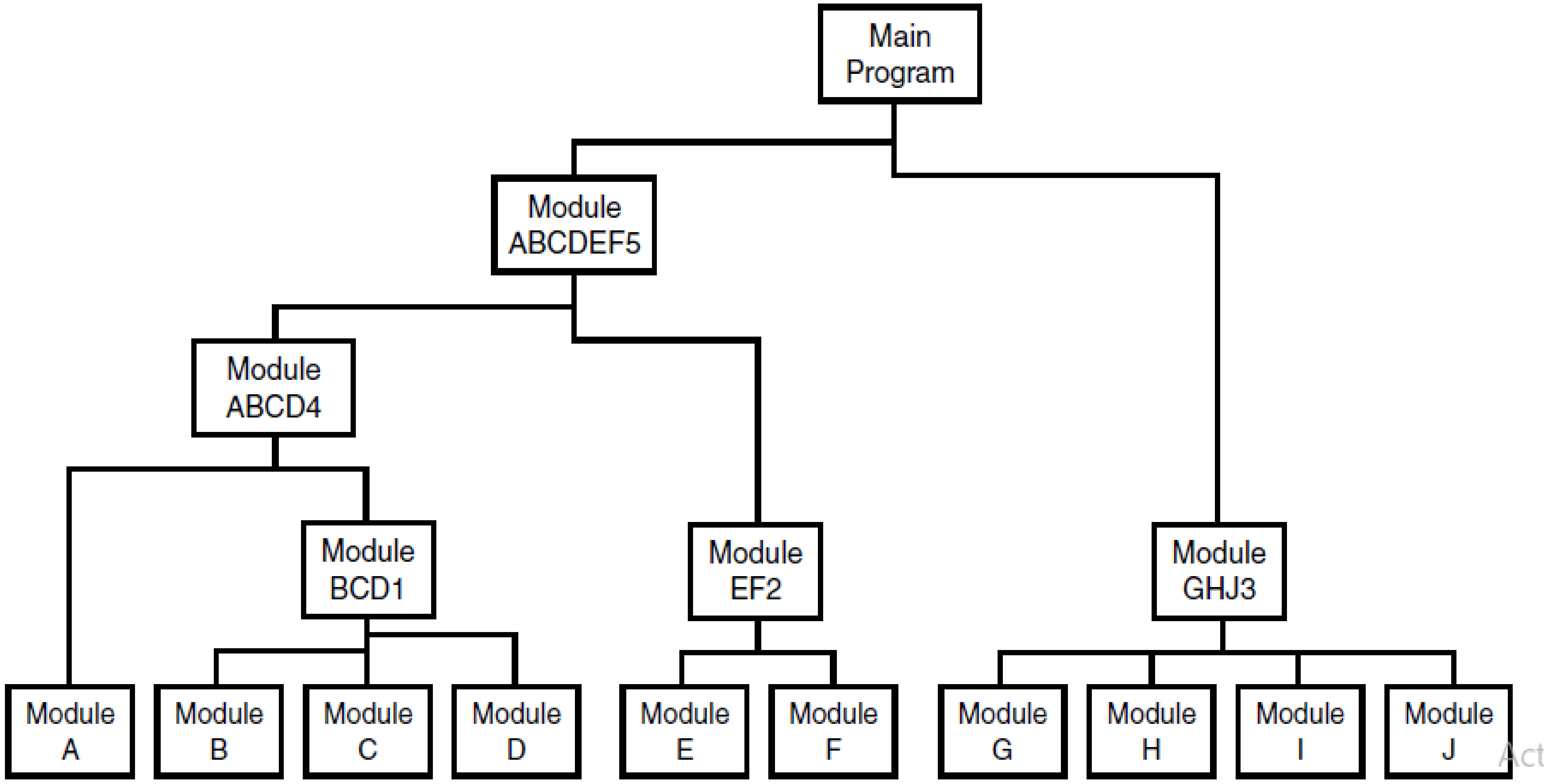
1. It’s difficult and sometimes impossible to figure out exactly what caused the problem.
2. Some bugs hide others. A test might fail.

Unit and Integration Testing

- Individual pieces of code are built up and tested separately, and then integrated and teste






Main Program
Module ABCDEF5
Module ABCD4
Module BCD1
Module EF2
Module GHJ3
Module A
Module B
Module C
Module D
Module E
Module F
Module G
Module H
Module I
Module J

UNIT TESTING

- **UNIT TESTING** is a type of software testing where individual units or components of a software are tested.

1. Unit Testing is done during the development.
2. A unit may be an individual function, method, procedure, module, or object.
3. Unit testing is a White Box testing technique that is usually performed by the developer.
4. Unit testing is first level of testing done before integration testing.

Cont..

- **Why Unit Testing?**

If proper unit testing is done in early development, then it saves time and money in the end.

- key reasons to perform unit testing

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use.

- Unit Testing is of two types

1. Manual
2. Automated

Cont..

- **Unit Testing Techniques**

1. Statement Coverage
2. Decision Coverage
3. Branch Coverage
4. Condition Coverage
5. Finite State Machine Coverage

integration testing & system testing

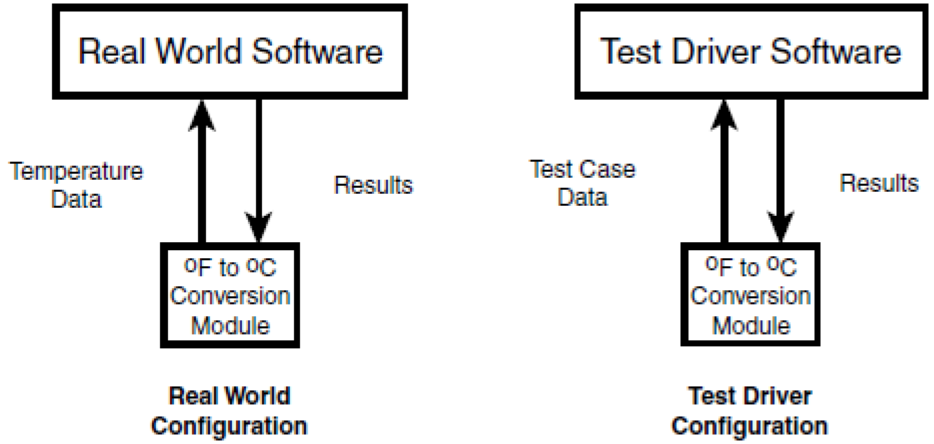
- As the units are tested and the low-level bugs are found and fixed, they are integrated and *integration testing* is performed against groups of modules.

- This process of incremental testing continues, putting together more and more pieces of the software until the entire product—or at least a major portion of it—is tested at once in a process called *system testing*.

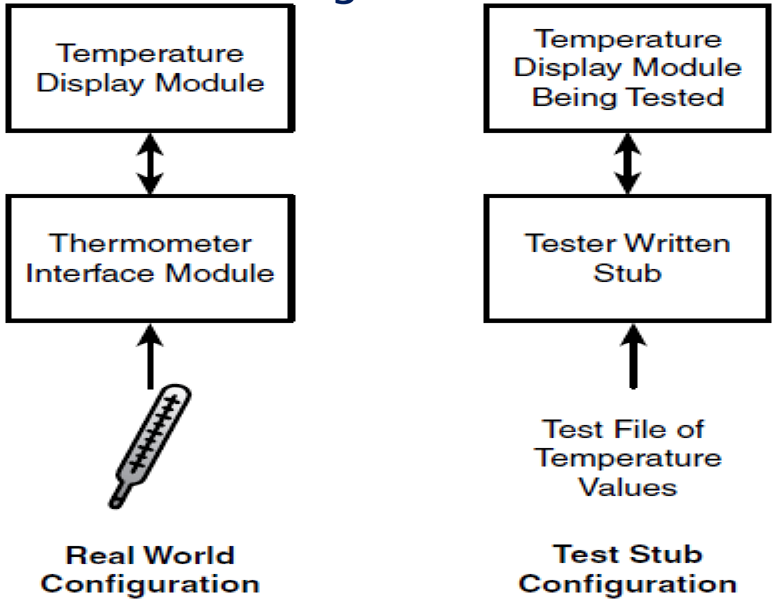
There are two approaches to this incremental testing: *bottom-up* and *top-down*

- Bottom-up testing you write your own modules, called *test drivers*, that exercise the modules you're testing.
- Top-down testing small piece of code called a *stub* that acts just like the interface module by feeding temperature values from a file directly to the display module.

A test driver can replace the real software and more efficiently test a low-level module.



A test stub sends test data up to the module being tested.


Temperature Display Module
Thermometer Interface Module
Real World Configuration
Temperature Display Module Being Tested
Tester Written Stub
Test File of Temperature Values
Test Stub Configuration

- **Data Coverage:** Consider the data first. Data includes all the variables, constants, arrays, data structures, keyboard and mouse input, files and screen input and output, and I/O to other devices such as modems, networks, and so on.

- **Data flow coverage :**involves tracking a piece of data completely through the software. A debugger and watch variables can help you trace a variable's values through a program.

- **Sub-Boundaries :**most common examples of sub-boundaries that can cause bugs, but every piece of software will have its own unique sub-boundaries, too. Here are a few more
- examples: A module that computes taxes might switch from using a data table to using a formula at a certain financial cut-off point.

- **Formulas and Equations :** Very often, formulas and equations are buried deep in the code and their presence or effect isn’t always obvious from the outside.

- **Error Forcing**
- If you’re running the software that you’re testing in a debugger, you don’t just have the ability to watch variables and see what values they hold—you can also force them to specific values.

- **Code-coverage analysis**: is a dynamic white-box testing technique because it requires you to have full access to the code to view what parts of the software you pass through when you run your test cases.
- The simplest form of code-coverage analysis is :compiler’s debugger

- **Code Coverage Methods**

1. Statement Coverage
2. Decision Coverage
3. Branch Coverage
4. Basis Path Testing

Program Statement or Line Coverage

- The most straightforward form of code coverage is called statement coverage or line coverage.

- Statement coverage:

1. It comes under white box testing.
2. This technique involves execution of all statements of the source code at least once.
3. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.

Branch Coverage

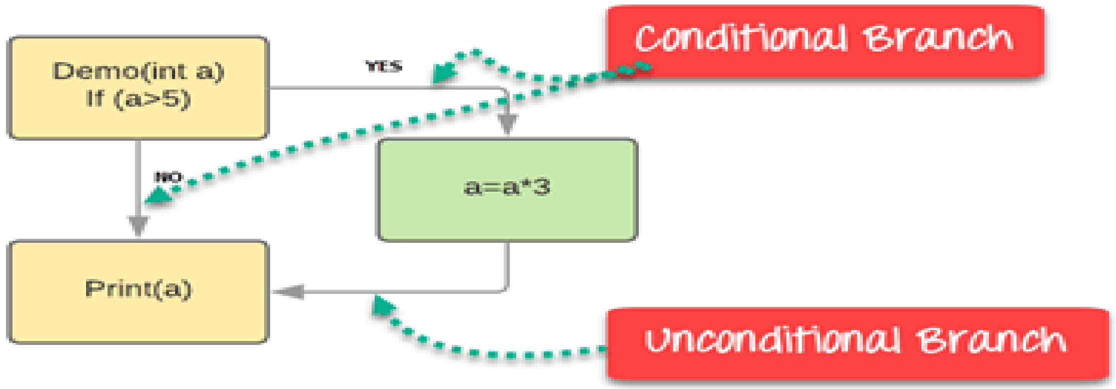
1. Attempting to cover all the paths in the software is called path testing.
2. The simplest form of path testing is called *branch coverage* testing.
3. **Branch Coverage** is a white box testing method in which every outcome from a code module(statement or loop) is tested.

$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

4. The purp : each decision condition from every branch is executed at least once.

Branch coverage Testing offers the following advantages

1. Allows you to validate-all the branches in the code.
2. Helps you to ensure that no branched lead to any abnormality of the program's operation.
3. Branch coverage method removes issues which happen because of statement coverage testing.
4. Allows you to find those areas which are not tested by other testing methods.
5. It allows overage
6. Branch c n expressions



Condition Coverage

- **Condition Coverage** or expression coverage is a testing method used to test and evaluate the variables or sub-expressions in the conditional statement.
- The goal of condition coverage is to check individual outcomes for each logic

$$\mathit{Condition\ Coverage} = \frac{\mathit{Number\ of\ Executed\ Operands}}{\mathit{Total\ Number\ of\ Operands}}$$

Condition coverage offers better sensitivity to the control flow than decision coverage.
Condition coverage does not give a guarantee about full decision coverage.

Basis path testing

- Basis path testing: A structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once.

- Cyclomatic Complexity: **Cyclomatic complexity** is a software metric used to indicate the **complexity** of a program.

- It is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976.

CALCULATION OF CYCLOMATIC COMPLEXITY

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

$M = E - N + P$

$M = 8 - 7 + 2$

$M = 3$

E = the number of edges of the graph

N = the number of nodes of the graph.

P = the number of connected components.

$M = E - N + 2$

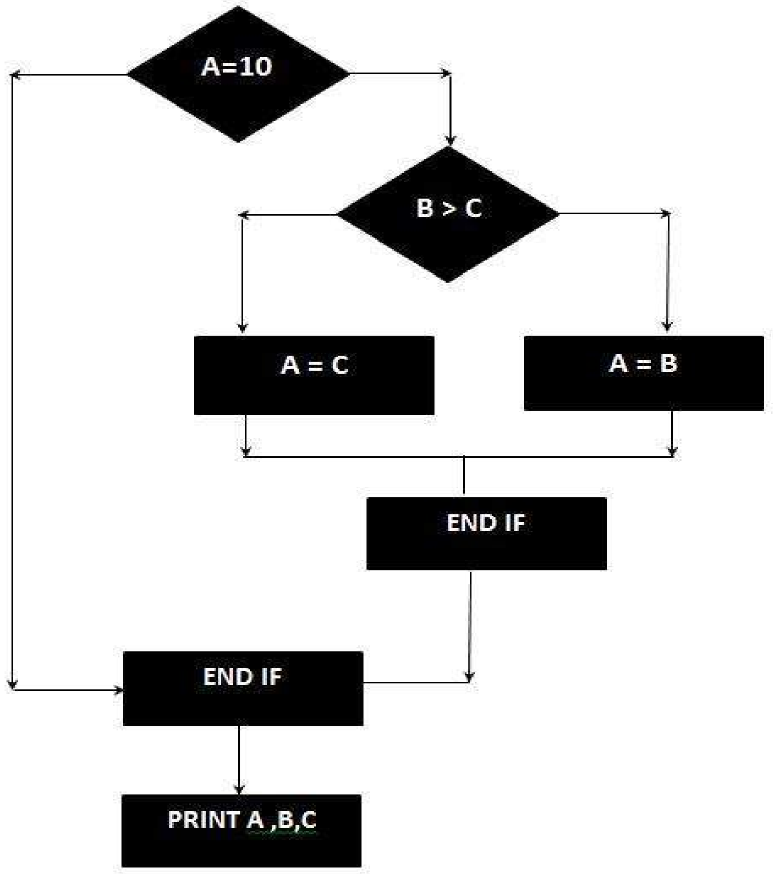
M=8-7=2

M=3

$M = D(Decision\ point\ ) + 1$

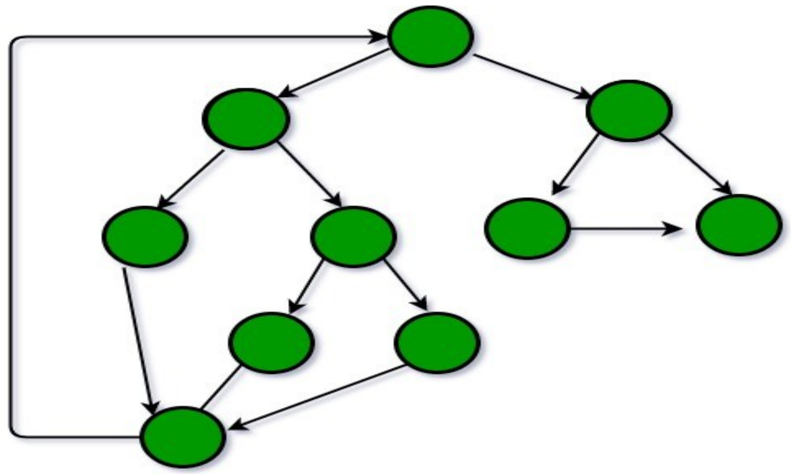
M=2+1

M=3



Example 2

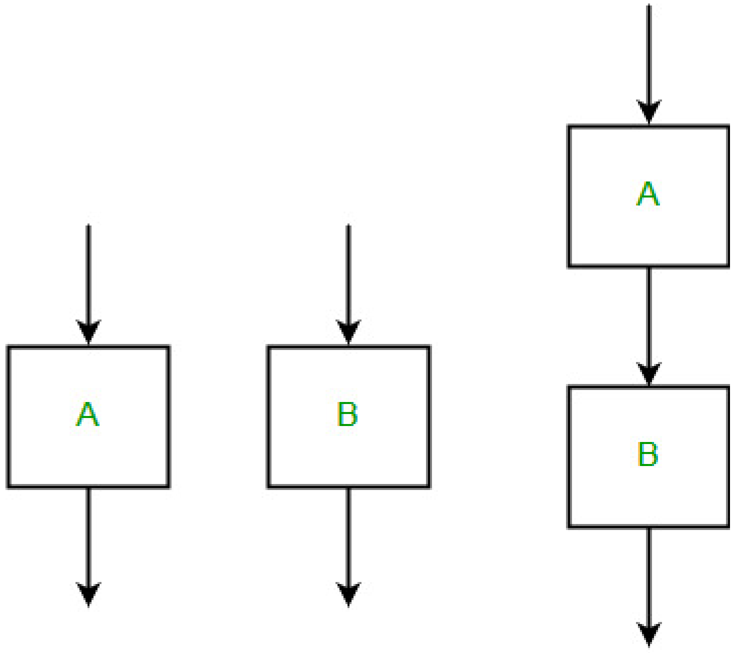
```
while (first <= last)
{
if (array [middle] < search)
    first = middle +1;
else if (array [middle] == search)
    found = True;
else last = middle – 1;
middle = (first + last)/2;
}
if (first < last) not Present = True;
```



```
 M = E − N + 2
M=13-10+2
= 5
```

Example 3

- The cyclomatic complexity of each of the modules A and B shown below is 10. What is the cyclomatic complexity of the sequential integration shown on the right hand side?



Number of decision points in A = 10 - 1 = 9
Number of decision points in B = 10 - 1 = 9

Cyclomatic Complexity of the integration
Number of decision points in A = 10 - 1 = 9
Number of decision points in B = 10 - 1 = 9

Cyclomatic Complexity of the integration = Number of decision points + 1
= (9 + 9) + 1
= 19n = Number of decision points + 1 = (9 + 9) + 1 = 19

- Consider the following program module:

```
int module1 (int x, int y)
{
 while (x! = y)
 {
 if (x > y)
x = x - y,
else y = y - x;
}
return x;
}
```

Cyclomatic complexity= D+1:2+1=3

Code Coverage v/s **Functional Coverage**

Code Coverage	**Functional Coverage**
Code coverage tells you how well the source code has been exercised by your test bench.	Functional coverage measures how well the functionality of the design has been covered by your test bench.
Never use a design specification	Use design specification
Done by developers	Done by Testers