Software Testing

→ It is an important discipline and consumes significant amount of effort and a proper strategy has to be implemented to carry out testing activities systematically and effectively.

Software Testing:- Testing of Software Product

↓

is a specialised discipline requiring unique skills.

What do you mean by Testing?

→ Testing is a process that demonstrates that errors are not present

→ The purpose of testing is to show that a program performs the intended functions correctly.

→ It is a process of establishing a connection/confidence that program is doing what it is supposed to do.

→ These definitions are opposite to what the testing actually does

Testing:- Testing is a process of executing a program with intent of finding errors.

Thus by testing we are increasing the reliability.

Why Should we Test?

→ Testing is an expensive activity yet launching Software without testing may lead to cost exponentially much higher than that of testing, Specially in the Systems where Human safety is involved

→ In Software life cycle the earlier the errors are detected and removed lower is the cost of their removal

What Should we Test → we should test the programs response to every possible input ie we should test the program for all valid or invalid inputs

→ we should test those areas where chances of getting a fault is maximum.

→ Organizations should develop strategies and techniques for effective testing.

↓

A strategy should be developed for testing small portion of program also to develop test cases for whole system.

Error:- People make error. Logical errors, Syntax errors or errors due to misunderstanding of specification

Mistake:- ~~When developers make errors they code~~ Synonym of Error

Bug :- When developers make mistakes during coding.

Fault:- is a representation of error, where representation is a mode of expression such as narrative text, DFD, ERD. Defect is a good synonym.

Failure:- occurs when fault executes. A particular fault may cause different failures depending upon how it has been exercised.

Software Testing

- Test case describes an input description and an expected output description.
- The set of test cases is called as test suite. Hence combination of test cases may generate a test suite.

Test Case ID	
Before execution	After execution
Purpose Precondition Input Outputs Post conditions Written by Date	Execution history: Result :- If fail any reason! Any other observation Any suggestion Run by Date

Test Case Template

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy conditions imposed at the start of that phase.

Validation is the process of evaluating a system or component during or at the end of development process to determine weather it specified the requirements

+

||

Software Testing.

Acceptance Testing : When software is developed for a specific customer

A series of test cases are conducted to enable the customers to validate all requirements. They are conducted by end user / customer and are of adhoc or well planned test.

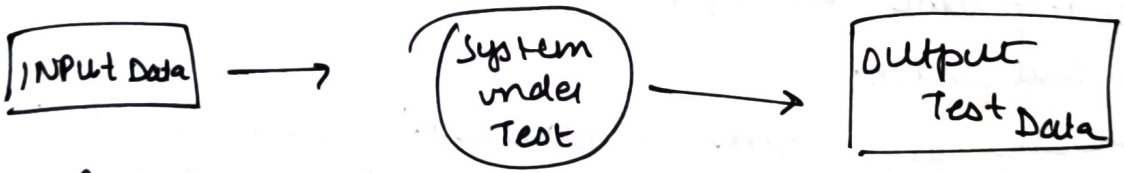
Alpha and Beta testing terms are used when the Software is developed for anonymus customers.

Alpha Tests are conducted at developers site by some potential customer. Testing is done in controlled enviornment Alpha testing starts when formal testing nears its completion

Beta Testing:- They are conducted by the end user at their site in a real world environment that is not controlled. Here there is no supervision of developer.

Types of Software Testing:-

① Black Box Testing: It is a method of testing that examines the functionality of the system without looking into the details of internal structure or working. In this we verify the functionality of the application based on requirement specification.
→ Performed by Test Engineers



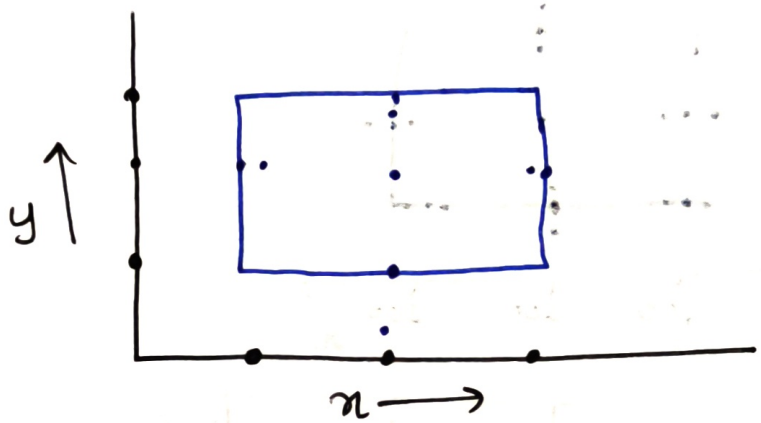
② White Box Testing:- It is a method to examine the inner workings of the system. Developers perform the whitebox testing and here we look for the code and the logic of the code is tested.
It is also called glass box or transparent testing.

Methods for Black Box Testing:-

1) Boundary Value Analysis:- It tests the system against the Boundary values of the various Input domains

Ex:- $a \le x \le b$, $c \le y \le d$ → Input Domain

Let x and y range from 100 to 300



Using Boundary value Analysis the Test cases generated are-

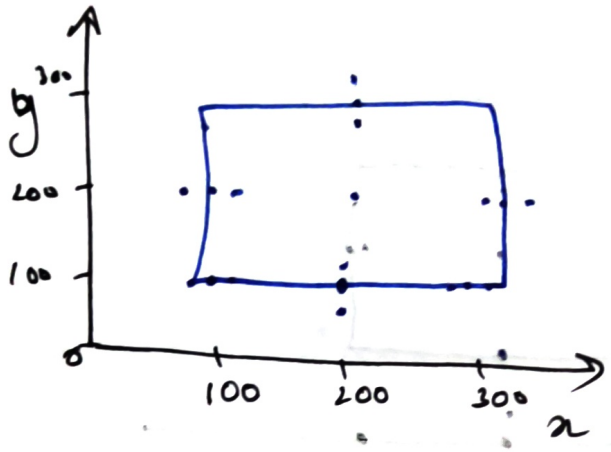
Test case	x	y	Expected output
1	200	100	
2	200	101	
3	200	200	
4	200	299	
5	200	300	
6	100	200	
7	101	200	
8	299	200	
9	300	200	

Boundary value Analysis gives us- $4(n)+1$ → Test cases (n → no. of input variables)

Robustness Testing:- An extension of Boundary value Analysis

→ Here we take values outside the boundary of input domain too.

Q $100 \leq x \leq 300$
$100 \leq y \leq 300$



Test cases	x	y	Expected output
1	200	99	
2	200	100	
3	200	101	
4	200	200	
5	200	299	
6	200	300	
7	200	301	
8	99	200	
9	100	200	
10	101	200	
11	299	200	
12	300	200	
13	301	200	

Robustness Testing gives : $6n+1$ Test cases

Worst Case Testing : Here we test the system for every possible input

→ Here we reject our single fault assumption theory and we what happens when one variable has extreme value.

→ Boundary Test cases are subset of worst case analysis.

$1 \leq x \leq 300$

$100 \leq y \leq 300$

Test case	x	y	Expected output
1	100	100	
2	100	101	
3	100	200	
4	100	299	
5	100	300	
6	101	100	
7	101	101	
8	101	200	
9	101	299	
10	101	300	
11	200	100	
12	200	101	
13	200	200	

Test case no.	x	y	Expected output
14	200	299	
15	200	300	
16	299	100	
17	299	101	
18	299	200	
19	299	299	
20	299	300	
21	300	100	
22	300	101	
23	300	200	
24	300	299	
25	300	300	

→ Worst case generates 5^n test cases for 'n' number of input variables.

→ Equivalence Class Testing :- Here the input domain is partioned into a finite number of equivalence classes that one can reasonably but not such assume, absolutely sure that test of a case from equivalence is a test for any other value.

Improvement

Two steps involved.

① Taking each input condition and partitioning it into valid and invalid classes.

② Generate the test case using the equivalence class covering all valid then invalid equivalence class.

→ Equivalence class must be written for output domain also.

Ex: for a program to identify the nature of quadratic equation. Identify equivalence class test cases for output and input domain.

$O_1 = \{ \langle a,b,c \rangle$: Not a quadratic Equation $(a=0) \}$

$O_2 = \{ \langle a,b,c \rangle$: Imaginary roots if $b^2 - 4ac < 0 \}$

$O_3 = \{ \langle a,b,c \rangle$: Equal roots if $b^2 - 4ac = 0 \}$

$O_4 = \{ \langle a,b,c \rangle$: Real roots if $b^2 - 4ac > 0 \}$

Test cases:-

Test cases	a	b	c	Expected output
1	0	1	2	Not a quadaratic eq
2	1	50	50	Real roots
3	50	100	50	Equal roots
4	50	50	50	Imaginary roots

for Input Domains:-

$I_1 = \{ a : a = 0 \}$

$I_2 = \{ a : a < 0 \}$

$I_3 = \{ a : 1 \leq a \leq 100 \}$

$I_4 : \{ a : a > 100 \}$

I_5 : {b > 100}

I_6 : {b < 0}

I_7 : {b <= 100}

I_8 : {0 <= c <= 100}

I_9 : {c < 0}

I_{10} : {c > 100}

Total Test cases = 10 + 4 = 14

Test case	a	b	c	Expected output
1	0	50	50	Not a quadratic Equation
2	-1	50	50	Invalid Input
3	50	50	50	Imaginary roots
4	101	50	50	Invalid Input
5	50	-1	50	Invalid Input
6	50	50	50	Imaginary roots
7	50	101	50	Invalid Input
8	50	50	0	Invalid Input
9	50	50	-1	Invalid Input
10	50	50	50	Imaginary roots.

Structural Testing:- Which test the internal structure of the program also called as whitebox testing.

→ **Path testing:-** It is a group of testing techniques based on judiciously selecting on a set of test path through the program. If the set of path is properly choosen, than it means we have achieved some test thoughness.

This involves:- generating path set that will cover every branch of program.

→ finding a set of test cases that will execute every

path in the path set.

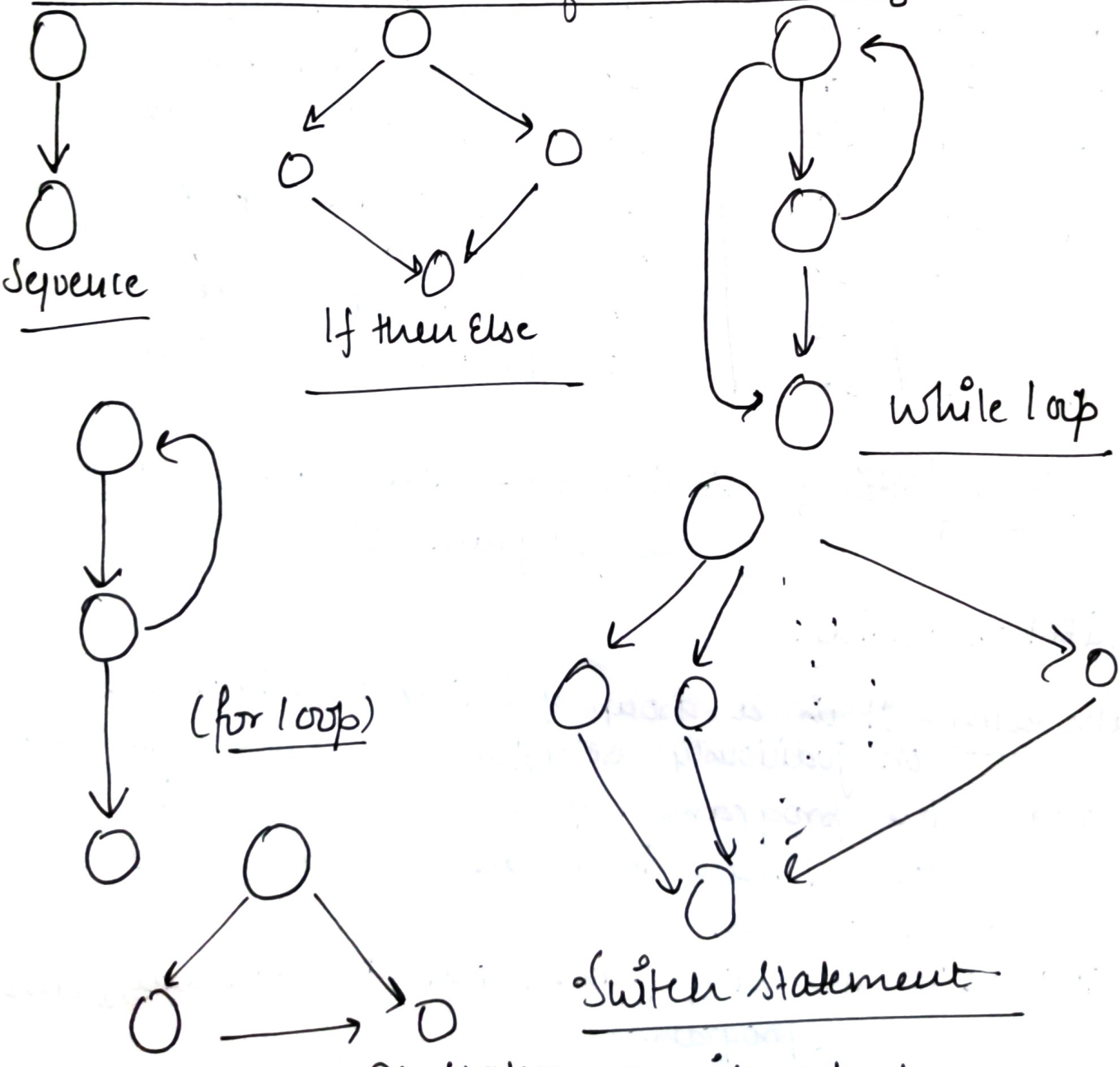
◇ Flow Graph:- The control flow of a program can be analysed using a graphical representation known as flow graph. The flowgraph is a directed graph where each node represent one or more procedural statements

→ directed edges represent the flow of control.

→ edge must terminate at the node.

→ Areas bounded by edges and nodes are called regions while counting the regions the outer area is counted as outer region.

Various representations of a flow graph:-



Cyclomatic Complexity :-

Cyclomatic Complexity measure the logical complexity of the code. It measures the number of linearly Independent paths through the program code

Various formulas:-

① Number of closed Regions + 1

② Number of predicate Nodes + 1

③ E - V + 2

E → No. of edges

V → Number of vertices.

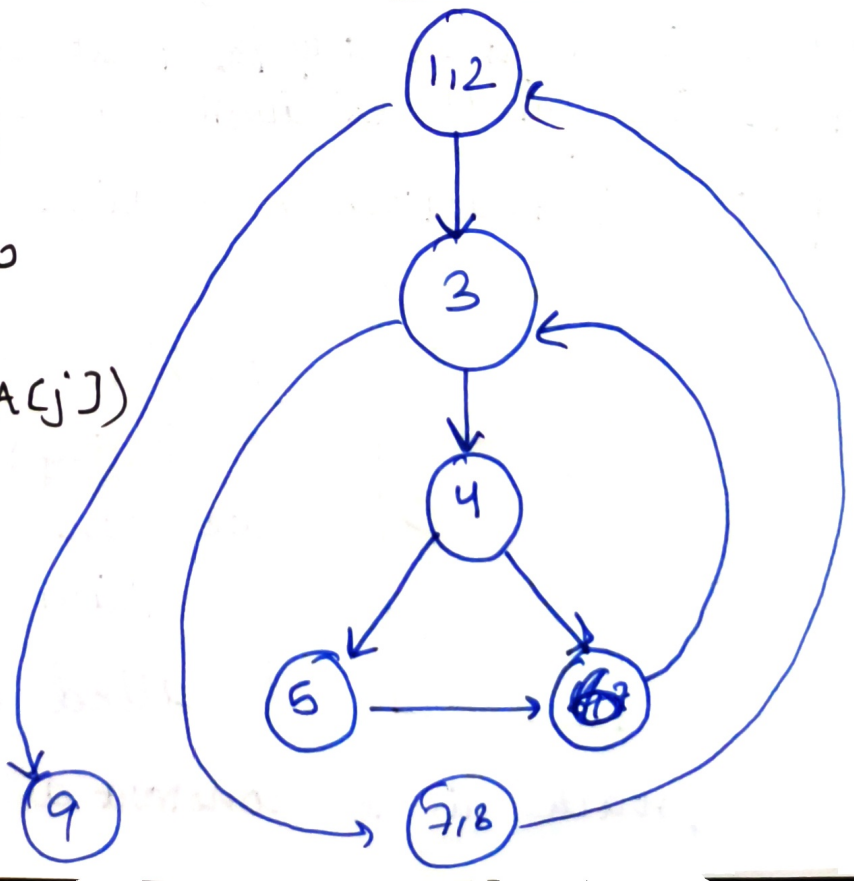
Set of Independent paths are known as basispath set

→ An independent path is a path through the program that introduces atleast one condition or new statement.

Example:- i=0
n=4

```
1 while (i<n-1) do
2 j=j+1
3 while (j<n) do
4 if A[i] < A[j]
5 swap (A[i], A[j])
6 endif;
7 end do.
8 j=j+1
9 end do;
```

Flow graph



Cyclomatic Complexity :- Number of closed regions + 1

$= 3 + 1$

$= 4$

→ Number of predicate nodes + 1

$= 3 + 1$

$= 4$

→ $E - V + 2$

$= 9 - 7 + 2$

$= 4$

Cyclomatic complexity is 4.

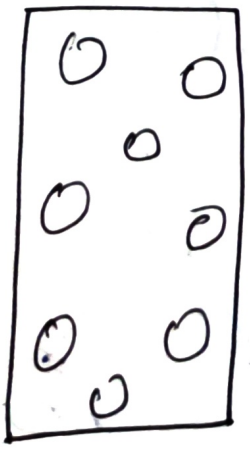
Levels of Testing :-

(i) Unit Testing :- When we test each module seperately

→ Each module is smaller in size so there is fairly easiness in locating errors.

→ Module is small enough that we can attempt to test it in fairly exhaustive manner.

→ Confusing interactions of multiple errors are eliminated.



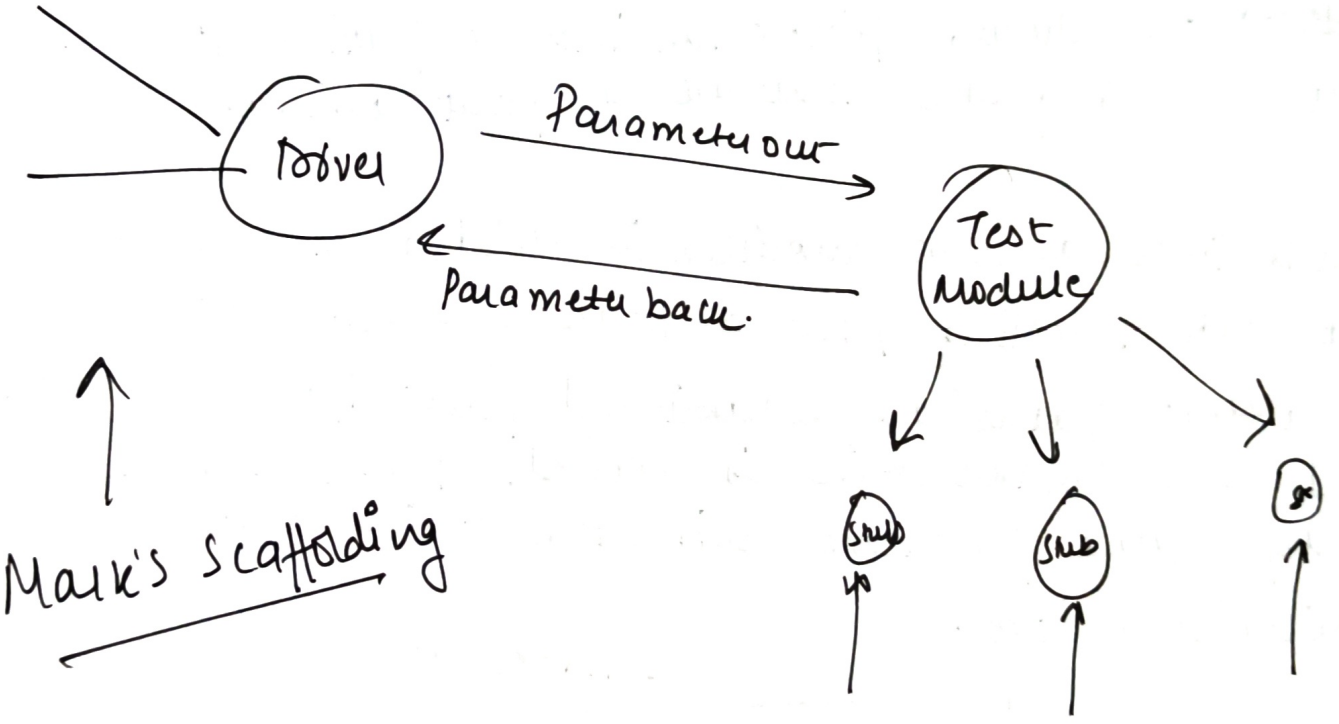
Problems with Unit Testing :- How can we run a module without anything to call it or to be called by it.

→ One approach is to construct an appropriate

driver routine to call it and stub to be called by it and to insert output statements in it

→ Stubs serve to replace the modules which are called by the modules to be tested. A stub or a dummy program uses the subordinate module's interface may do minimal data manipulation /verification of entry and return.

→ This overhead is called scaffolding representing effort that is import to testing.



Integration Testing :- The purpose of unit testing is to determine each independent module is correctly implemented. This also helps in testing the interface b/w the various modules : weather parameters match on both sides, permissible range, meaning and utilization.

Types:

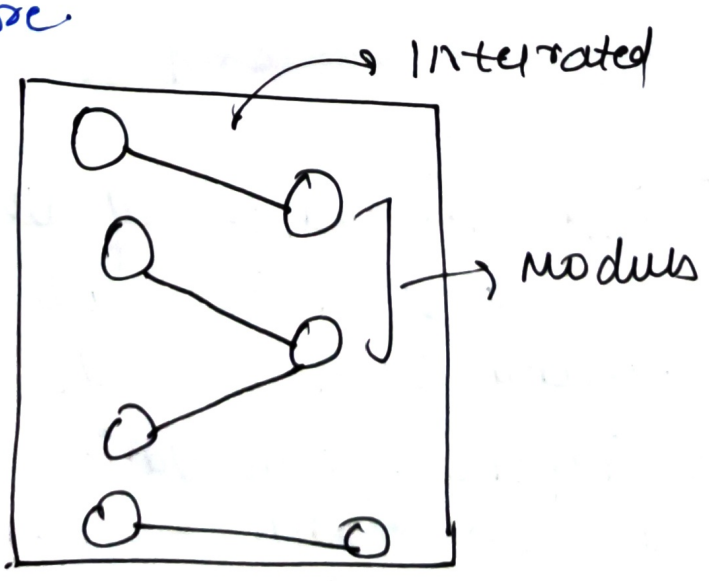
Top down:- Proceeds down in hierarchy adding one module at a time until entire tree is integrated. Thus eliminating the need of drivers. It should be applied when control and interface are a part of complex sqtr.

Bottoms up:- works similarly for bottom and has no need of stubs. It is applied if machine interface and performance is of special interest.

Sandwich:- It runs top to bottom concurrently with bottom to up uniting somewhere in middle.

→ Integration should follow the lines of putting together those subsystems that are of great concern.

→ Each time a new module is added as a part of Integration testing, the Sqtware changes new data flow paths are established, new I/o may occur and new control logic is evoked. These changes may cause problems with the units that worked flawlessly before.



System Testing:- Software is one large component of computer based system

→ Software is incorporated with other components i.e Hardware and software is run on different hardware configuration.

→ Software is run on different Hardware setup including full range of memory, processor, OS and peripherals

→ Response time under various conditions is recorded to achieve specific level of performance.

→ Beizer gives some guidelines for choosing test cases during system testing.

→ We should test various attributes of software during System Testing. These represents the Operational Correctness of product

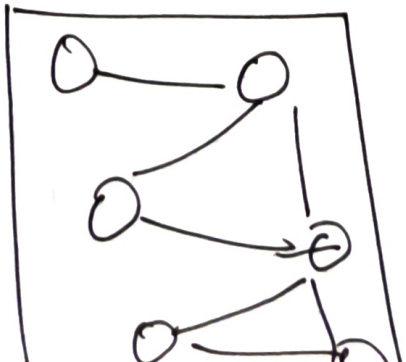
Usable Is the product clear, convinent and predictable?

Secure Is acces to sensitive data restricted to those with authorization

Compatible Will the product work correctly in conjunction with existing data, software and procedures?

Dependable Do methods of safeguards and methods for recovery exist.

Documented All manuals are complete, correct and understandable?



→ System under test

Validation Testing:- It refers to test software as a complete product

→ Done after unit & Integrated testing.

→ Alpha, Beta and acceptance testing are nothing but involving customer during testing.

→ Validation testing improves the quality of software product in terms of functional capabilities and quality attributes such as accuracy, completeness, testability, portability, maintainability etc.

IEEE:- has developed a standard named "IEEE guide for software verification and validation"

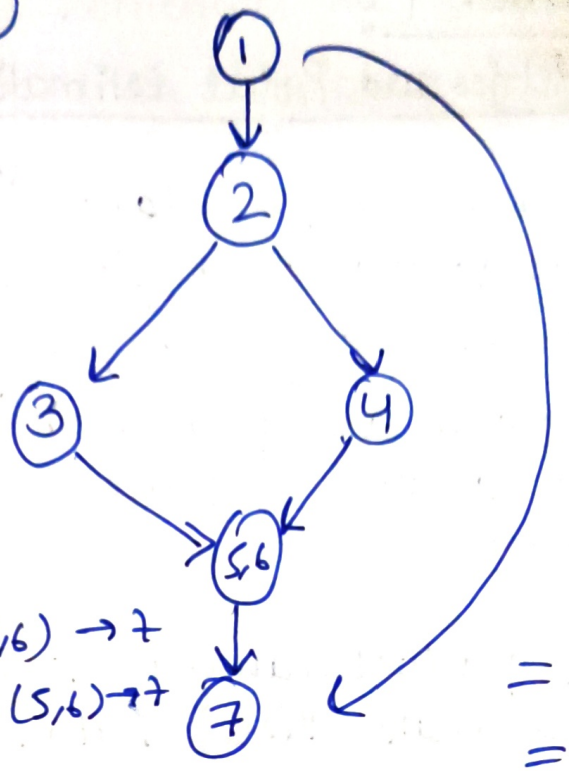
Examples on flow graph:-

Question:- for the following code snippets draw flowgraph and calculate cyclomatic complexity

```
1  If A=4
2  Then If  B>C
3       Then A=B
4       Else  B=C
5     EndIf
6   EndIf
7   Print A
```

②
```
begin int x, y power
1  float z;
2  input (x, y);
3  if (y<0)
4  power= -y
5  else power=y
6  ~~Power~~ endIf
7  z=1
8  while (power!=0) do
9    z=z*x
10   power= power-1
11 end while
12 If (y<0)  z=1/z;
13 endIf
14 output (z);
15 End
```

for code (1)



Cyclomatic complexity = ~~3~~+1
= ~~4~~ 3

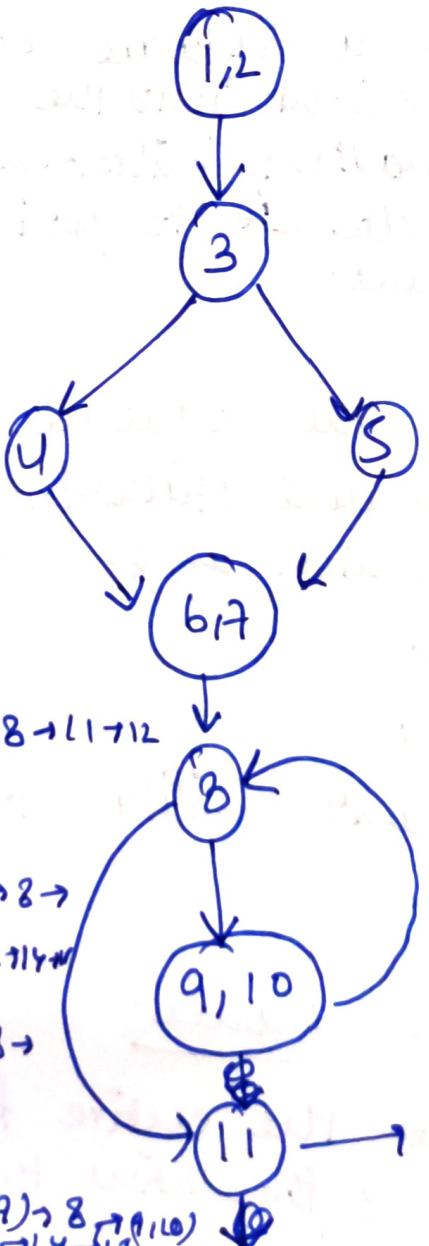
Path set:
1 → 7

1 → 2 → 3 → (5,6) → 7
1 → 2 → ~~3~~ 4 → (5,6) → 7

= V + 1
= 2 + 1 = 3

= 7 − 6 + 2
= 3

for code Snippet (2)



Cyclomatic complexity
⇒ 3+1
= 4

⇒ V + 1
= 3 + 1 = 4

⇒ 12 − 12 + 2
= 4

Path set:

1) (1,2) → 3 → 4 → (6,7) → 8 → 11 → 12
→ 13 → 14 → 15

2) (1,2) → 3 → 4 → (6,7) → 8 →
(9,10) → 8 → 11 → 12 → 14 → 15

3) (1,2) → 3 → ~~5~~ → (6,7) → 8 →
11 → 12 → 14 → 15

4) (1,2) → 3 → 5 → (6,7) → 8 → (9,10)
→ 8 → 11 → 12 → 14 → 15