Spring-2020

1. a) Give a comparison 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.
Testing is the display of errors.	Debugging is a deductive process.
Testing is planned, designed and scheduled.	Debugging cannot have such constraints.
Testing can be done without design knowledge.	Debugging is impossible with detailed design knowledge.
It can be done by an outsider.	It must be done by an insider.
Testing can be manual or automated.	Debugging is always manual as it can't be automated.

1. b) Distinguish between Stress Testing & Load Testing. Write down the principles of software testing.

Load Testing	Stress Testing
Load Testing is performed to test the performance of the system or software application under extreme load.	Stress Testing is performed to test the robustness of the system or software application under extreme load.
In load testing load limit is the threshold of a break.	In stress testing load limit is above the threshold of a break.
In load testing, the performance of the software is tested under multiple number of users.	In stress testing, the performance is tested under varying data amounts.
Huge number of users.	Too much users and too much data.
Load testing is performed to find out the upper limit of the system or application.	Stress testing is performed to find the behavior of the system under pressure.
The factor tested during load testing is *performance*.	The factor tested during stress testing is *robustness* and *stability*.
Load testing determines the operating capacity of a system or application.	Stress testing ensures the system security.

Principles of Software Testing:

1. Testing shows the presence of defects
2. Exhaustive testing is impossible
3. Early testing reduces the cost
4. Defect clustering
5. Pesticide paradox

6. Testing is context dependent
7. Absence of error – fallacy

1. c) Define Coverage Criterion. With necessary diagram briefly describe the MDTD activities.

Coverage criteria gives structured, practical ways to search the input space. Satisfying a coverage criterion gives a tester some amount of confidence in two crucial goals:

Search the input space thoroughly

Not much overlap in the tests

MDTD Activities:

1. Test Design – Test design can be further broken into two categories:
   - Criteria-based: Design test values to satisfy coverage criteria or other engineering goal. It requires the knowledge of –
     - Discrete math
     - Programming
     - Testing

     It also requires a traditional CS degree.
   - Human-based: Design test values based on domain knowledge of the program and human knowledge of testing. Criteria based approaches can be blind to special situations. It requires knowledge of –
     - Domain
     - Testing
     - UI

     It requires no traditional CS.
2. Test Automation – Embed test values into executable scripts. Its slightly less technical, requires knowledge of programming, but very little theory. It often requires observability and controllability to solve difficult problems.
3. Test Execution – Run tests on the software and record the results. Its easy and can be automated. Requires basic computer skills.
4. Test Evaluation – Evaluate results of testing and report to developers. Requires knowledge of:
   - Domain
   - Testing
   - UI
   - Psychology

   It does not require any traditional CS.

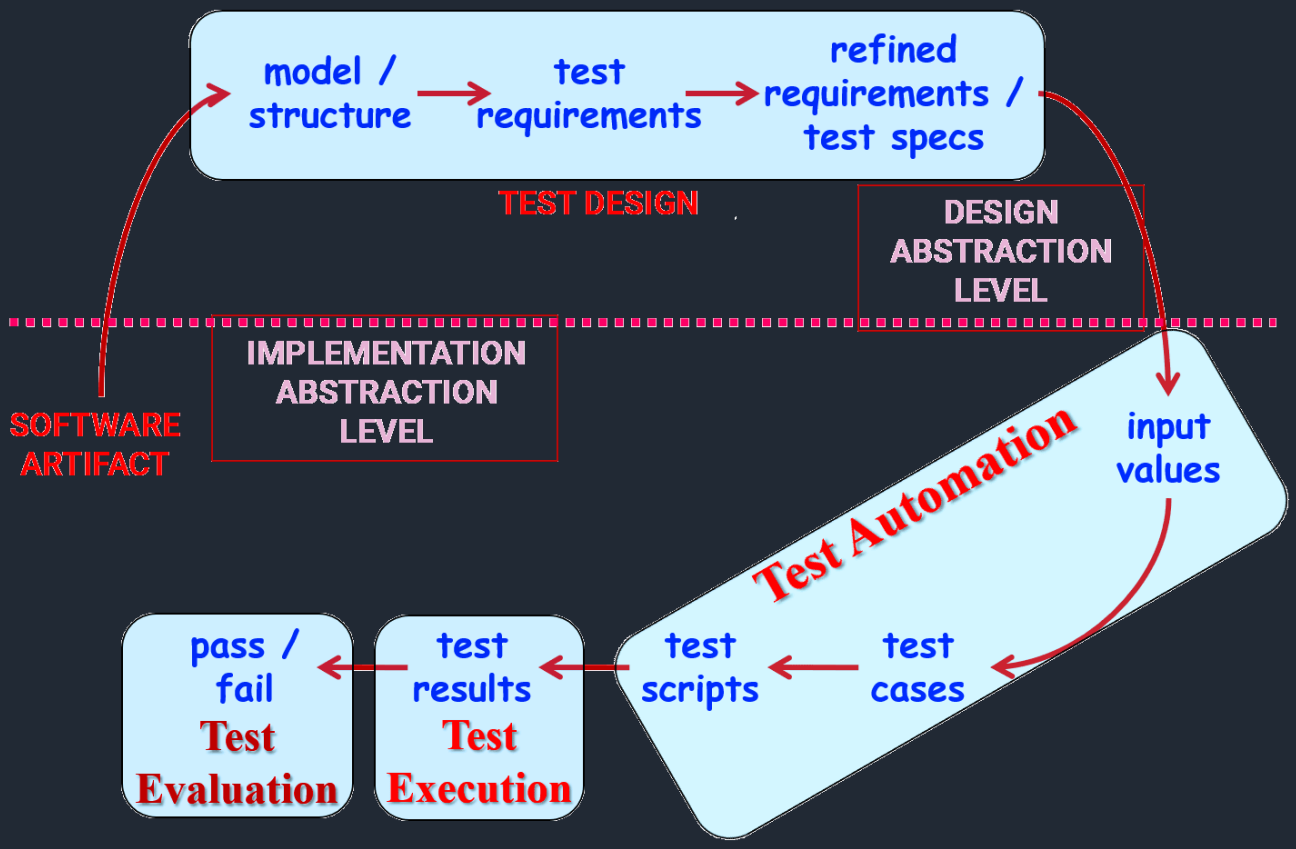


Diagram: MDTD Activities

2. Consider the following information about a graph and answer each of the followings:

$N = \{1,2,3,4,5,6,8\}$

$N_0 = \{1\}$

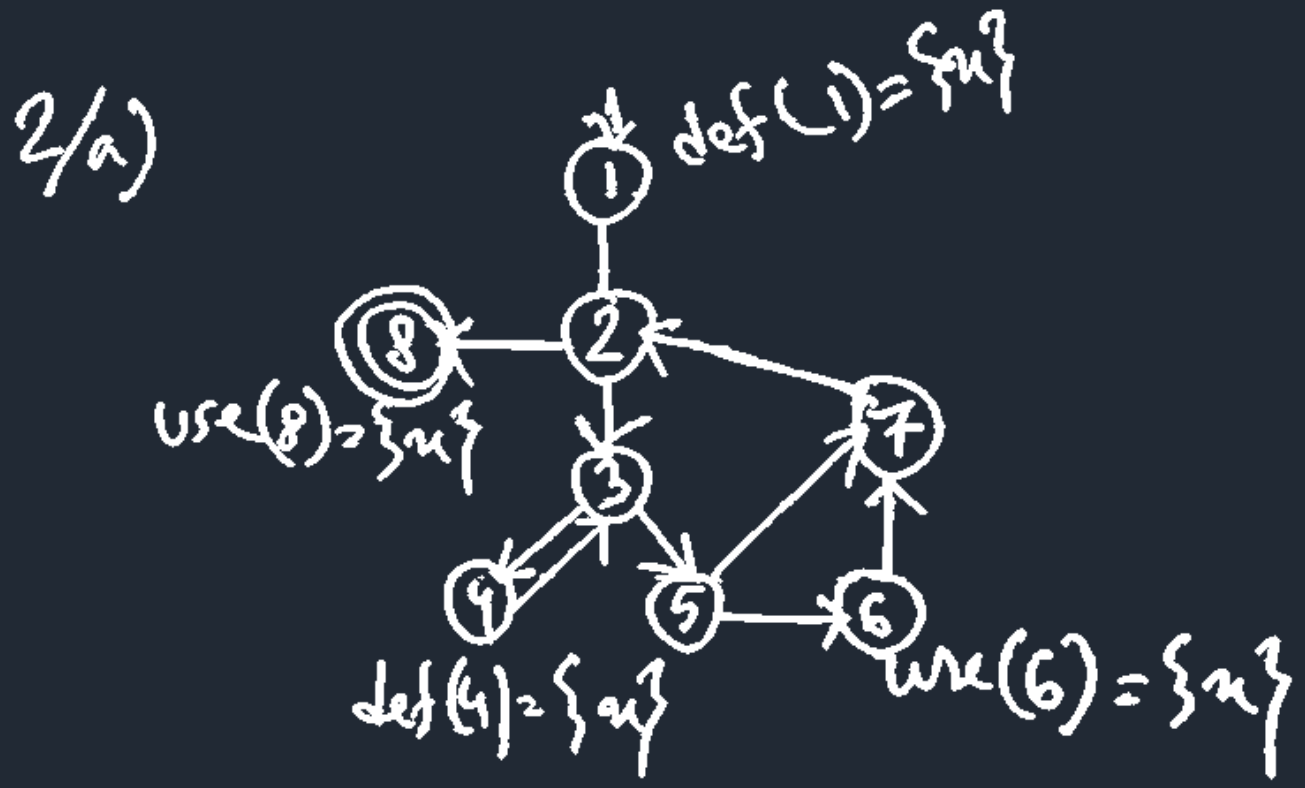
$N_f = \{8\}$

$E = \{(1,2), (2,3), (2,8), (3,4), (3,5), (4,3), (5,6), (5,7), (6,7), (7,2)\}$

$def(1) = def(4) = use(6) = use(8) = \{x\}$

a) Draw the graph

Graph is given below:



b) List all the du-paths with respect to x.

All du-paths with respect to x are listed below:

du-pair	du-paths
(1,8)	[1,2,8] [1,2,3,5,7,2,8]
(1,6)	[1,2,3,5,6]
(1,(6,8))	[1,2,3,5,6,7,2,8]
(4,6)	[4,3,5,6]
(4,8)	[4,3,5,7,2,8]
(4,(6,8))	[4,3,5,6,7,2,8] [4,3,5,7,2,3,5,6,7,2,8]

c) List a minimal test set that satisfies all uses coverage with respect to x.

Minimal test set is given below for AUC:

[1,2,3,5,6,7,2,8]

[1,2,3,4,3,5,7,2,3,5,6,7,2,8]

d) List a minimal test set that satisfies all DU-paths coverage with respect to x.

Minimal test set for ADUPC:

[1,2,8]

[1,2,3,5,7,2,8]

[1,2,3,5,6,7,2,8]

[1,2,3,4,3,5,7,2,8]

[1,2,3,4,3,5,6,7,2,8]

[1,2,3,4,3,5,7,2,3,5,6,7,2,8]

3. a) Define predicate and clause.

A predicate is an expression that evaluates to a Boolean value. It can contain:

- Boolean variables
- Non-Boolean variables that contain >, <. ==, >=, <=, !=
- Boolean function calls

Internal structure is created by logical operators

- ¬ the negation operator
- $\wedge$ the and operator
- $\vee$ the or operator
- $\rightarrow$ the implication operator
- $\oplus$ the exclusive or operator
- $\leftrightarrow$ the equivalence operator

A clause is a predicate with no logical operators. For example, (a < b) $\vee$ f (z) $\wedge$ D $\wedge$ (m >= n*o) has four clauses:

- (a < b) – relational expression
- f (z) – Boolean-valued function
- D – Boolean variable
- (m >= n*o) – relational expression

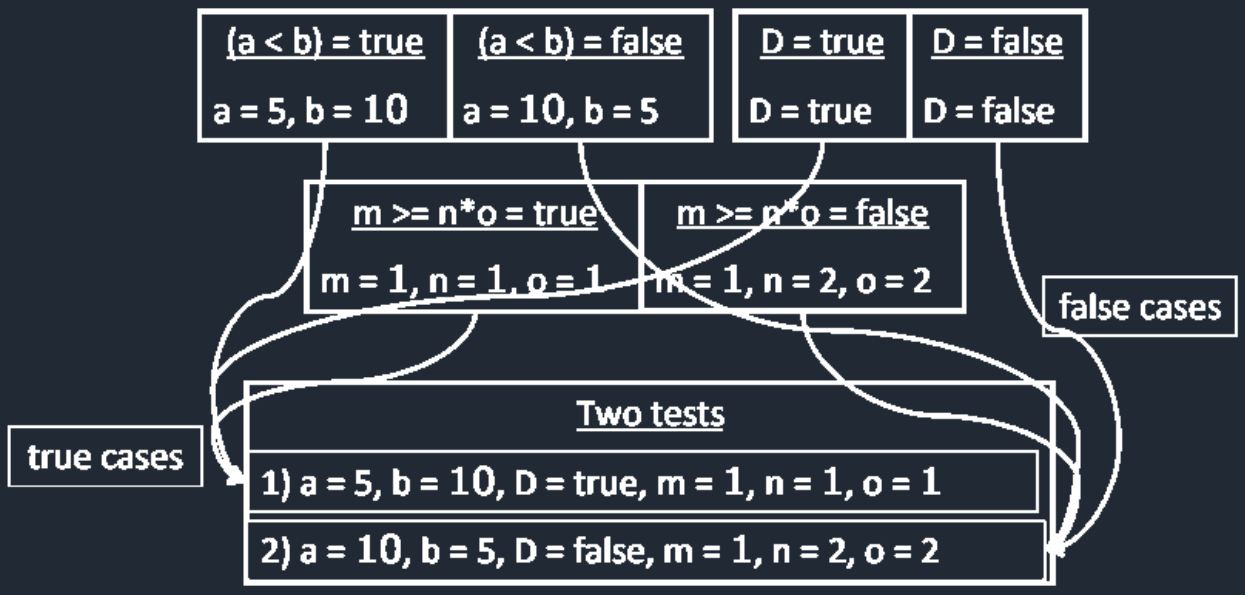
3. b) Consider the logic expression, p = $(((a < b) \vee D) \wedge (m >= n * o)$ and answer the followings:

i) List down the clauses

Clauses:

- (a<b) – relational expression
- D – Boolean variable
- m>=n*o – relational expression

ii) Determine any test cases for clause coverage.



3. c) Define predicate coverage and combinational coverage.

Predicate Coverage (PC): For each p in P, TR contains two requirements: p evaluates to true, and p evaluates to false.
Clause Coverage (CC): For each c in C, TR contains two requirements: c evaluates to true, and c evaluates to false.

3. d) Determine the CACC and RACC pairs of the clauses for the following logic expression:
$p = (\neg a \wedge \neg b) \vee (a \wedge \neg c) \vee (\neg a \wedge c)$

SL	a	b	c	Predicate	Pa	Pb	Pc
1	TRUE	TRUE	TRUE	FALSE			
2	TRUE	TRUE	FALSE	TRUE			
3	TRUE	FALSE	TRUE	FALSE			
4	TRUE	FALSE	FALSE	TRUE			
5	FALSE	TRUE	TRUE	TRUE			
6	FALSE	TRUE	FALSE	FALSE			
7	FALSE	FALSE	TRUE	TRUE			
8	FALSE	FALSE	FALSE	TRUE			

Major clause:

Pa: b=true or c=true

CACC can be satisfied by choosing any of the rows 1,2,3 AND any of the rows 5,6,7. A total of 9 pairs.

RACC can be satisfied by row pairs (1,5), (2,6), (3,7). A total of 3 pairs.

4. a) Assume that, while doing ISP we found three characteristics $\{A, B, C\}$ and each of the characteristics are partitioned into blocks of different sizes $\{(A1, A2), (B1, B2, B3), (C1, C2, C3, C4)\}$. Now, answer each of the following questions:

i) How many test cases we will get for all combination coverage?

2*3*4 = 24

ii) How many test cases we will get for pair-wise coverage?

3*4=12

iii) How many test cases we will get for base choice coverage?

1+(2-1)+(3-1)+(4-1)=1+1+2+3=7

4. b) Define input domain.

Input domain is the set of all possible inputs to a program.

Consider the following code segment:

```
public boolean findElement (List list, Object element) // Effects: if list or element is null
throw NullPointerException // else return true if element is in the list, false otherwise
```

Now, give an example of partitioning scheme that will satisfy the following characteristic constraints for the above code snippet and highlight the criteria:

i) A block from one characteristic cannot be combined with a specific block from another.

ii) A block from one characteristic can ONLY BE combined with a specific block form another characteristic.

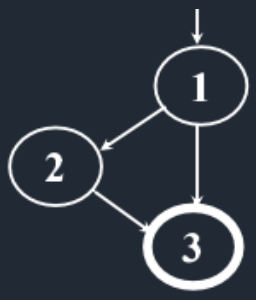
Answer:

Characteristic	Block 1	Block 2	Block 3	Block 4
A : length and contents	One element	More than one, unsorted	More than one, sorted	More than one, all identical
B : match	element not found	element found once	element found more than once	
Invalid combinations : (A1, B3), (A4, B2)				

4. c) Define each of the followings with an appropriate example:

I. **Node Coverage**

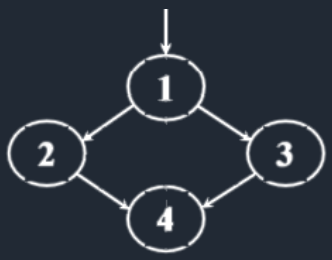
Node Coverage (NC): Test set T satisfies node coverage on graph G if and only if, for every syntactically reachable node n in N, there is some path p in path(T) such that p visits n.
Considering the following graph, TR & TP for NC would be:
TR = {1,2,3}
TP = [1,2,3]



II. **Prime Path**

Prime path is a simple path that does not appear as a proper sub-path of any other simple path.

Prime paths = [2,4,1,2], [2,4,1,3], [1,3,4,1], [1,2,4,1], [3,4,1,2], [4,1,3,4], [4,1,2,4], [3,4,1,3]



III. **Test Path**

Test path is a path that starts at an initial node and ends at a final node. It represents the execution of test cases where:

- Some test paths can be executed by many tests
- Some test paths cannot be executed by any tests

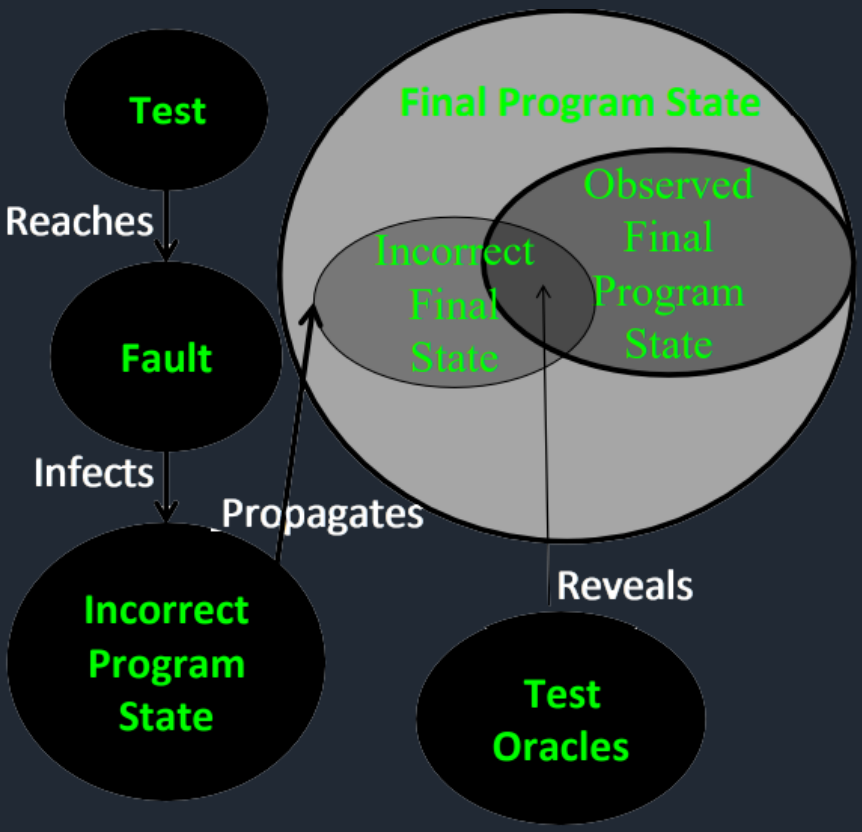
For example, TP for NC of the 4.c.i. is: TP = [1,2,3]

IV. **T-wise Coverage**

A value from each block for each group of t characteristics must be combined. For example, if there are 3 blocks with 3 characteristics, and we specify the 't' to be 3, then t-wise coverage would be = 3*3*3 = 27.

Fall-2020

1. a) Draw the block diagram of RIPR model.



1. b) Write down the principles of Software Testing.

Principles of Software Testing:

1. Testing shows the presence of defects
2. Exhaustive testing is impossible
3. Early testing reduces the cost
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of failure

1. c) Define each of the followings:

i) Beta Testing

Beta testing is **a type of user acceptance testing where the product team gives a nearly finished product to a group of target users to evaluate product performance in the real world**.

ii) Negative Testing

A Negative Testing technique is performed using incorrect data, invalid data or input where testers having the mindset of “attitude to break”. Using Negative Testing they validate that if system or application breaks.

iii) Load Testing

Load testing is a type of non-functional testing where the objective is to check how much load or maximum workload a system can handle without any performance degradation. It helps to find the maximum capacity of the system under specific load.

1. d) List down the activities of MDTD and draw the block diagram of MDTD Steps.

Activities of MDTD:

1. Test Design
    a. Criteria-based
    b. Human-based
2. Test Automation
3. Test Execution
4. Test Evaluation

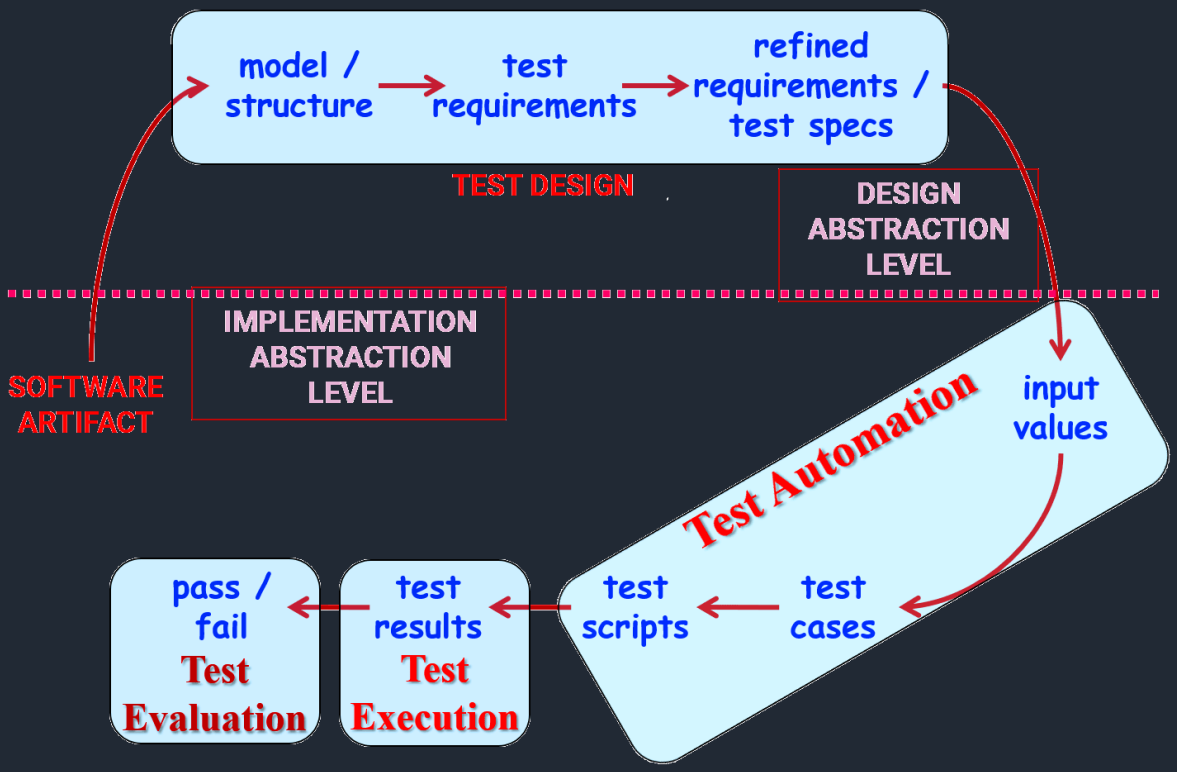


Diagram: MDTD Activities

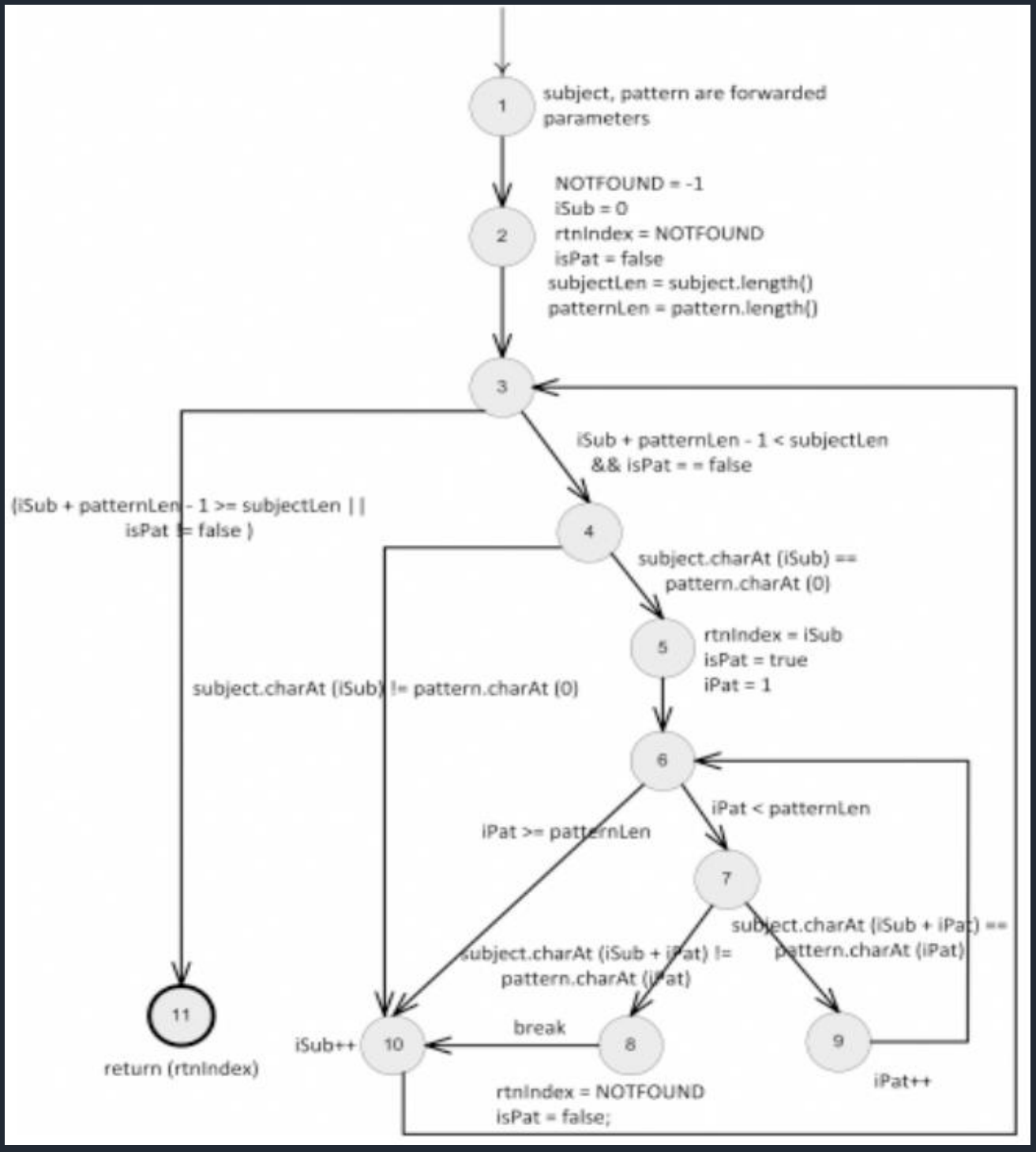
2. a) Define def-clear path and simple path.

A path from l_i to l_j is def-clear with respect to variable v, if v is not given another value on any of the nodes or edges in the path.

A path from node n_i to n_j is simple if no node appears more than once, except possibly the first and last nodes are the same.

2. b) Draw the dataflow diagram graph of Code Segment-I with proper annotations.

Dataflow graph:



2. c) Write down the edge pair coverage criteria of the derived graph of Code Segment-I.

EPC Criteria:

TR= {[1,2,3], [2,3,11], [2,3,4], [3,4,5], [3,4,10], [4,10,3], [4,5,6], [5,6,10], [5,6,7], [6,10,3], [6,7,8], [6,7,9], [7,8,10], [7,9,6], [8,10,3], [9,6,7], [9,6,10], [10,3,4], [10,3,11]}

2. d) Determine the du-paths for the variables isPat, rtnIndex, iSub of Code Segment-I.

DU-paths:

variable	du-path set	du-paths	prefix?
NOTFOUND	du (2, NOTFOUND)	[2,3,4,5,6,7,8]	
rtnIndex	du (2, rtnIndex)	[2,3,11]	
	du (5, rtnIndex)	[5,6,10,3,11]	
	du (8, rtnIndex)	[8,10,3,11]	
iSub	du (2, iSub)	[2,3,4]	Yes
		[2,3,4,5]	Yes
		[2,3,4,5,6,7,8]	Yes
		[2,3,4,5,6,7,9]	
		[2,3,4,5,6,10]	
		[2,3,4,5,6,7,8,10]	
		[2,3,4,10]	
		[2,3,11]	
	du (10, iSub)	[10,3,4]	Yes
		[10,3,4,5]	Yes
		[10,3,4,5,6,7,8]	Yes
		[10,3,4,5,6,7,9]	
		[10,3,4,5,6,10]	
		[10,3,4,5,6,7,8,10]	
		[10,3,4,10]	
		[10,3,11]	
isPat	du (2, isPat)	[2,3,4]	
		[2,3,11]	
	du (5, isPat)	[5,6,10,3,4]	
		[5,6,10,3,11]	
	du (8, isPat)	[8,10,3,4]	
		[8,10,3,11]	

3. a) Does predicate coverage subsumes clause coverage? Explain with example.

When predicates come from conditions on edges, this is equivalent to edge coverage. But PC does not subsume clause coverage. Let us consider the following example:

((a < b) ∨ D) ∧ (m >= n*o)

predicate coverage

Predicate = true

a = 5, b = 10, D = true, m = 1, n = 1, o = 1

= (5 < 10) ∨ true ∧ (1 >= 1*1)

= true ∨ true ∧ TRUE

= true

Predicate = false

a = 10, b = 5, D = false, m = 1, n = 1, o = 1

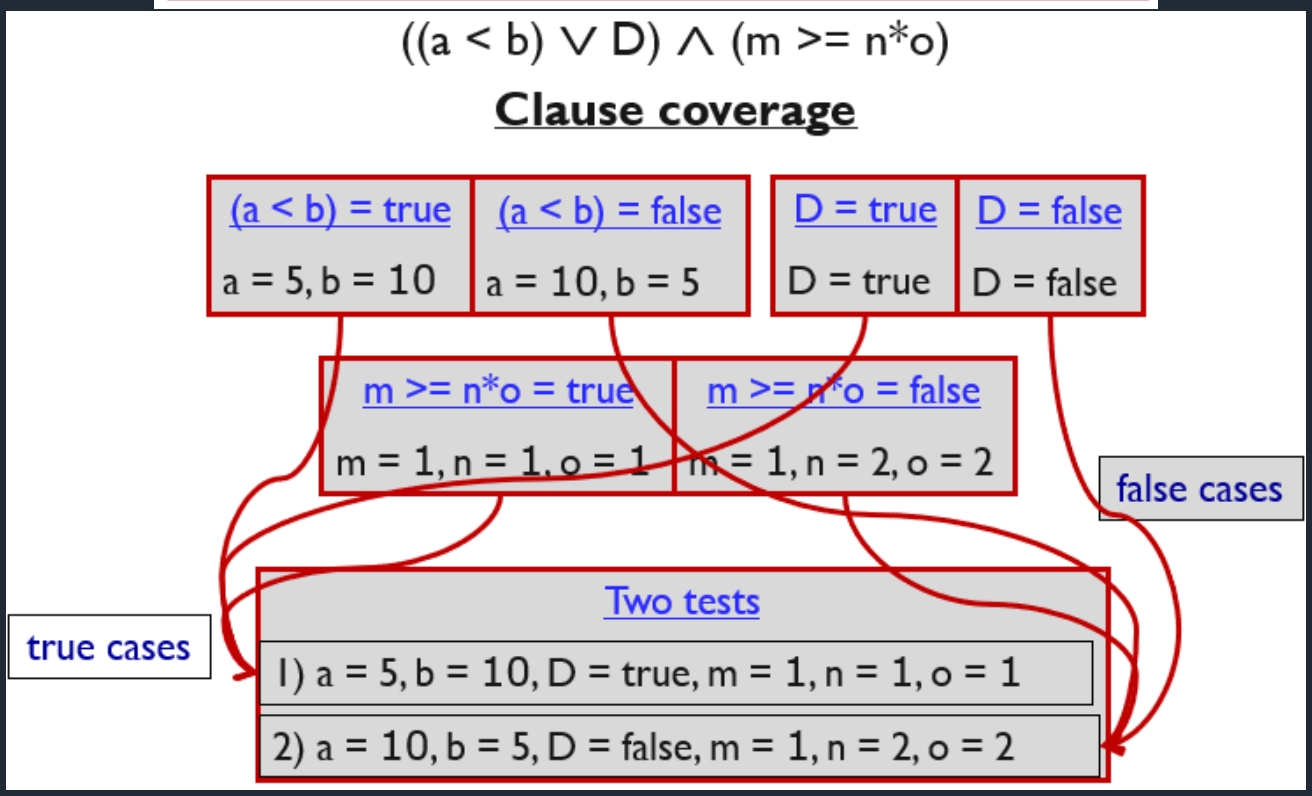
= (10 < 5) ∨ false ∧ (1 >= 1*1)

= false ∨ false ∧ TRUE

= false

((a < b) ∨ D) ∧ (m >= n*o)

Clause coverage



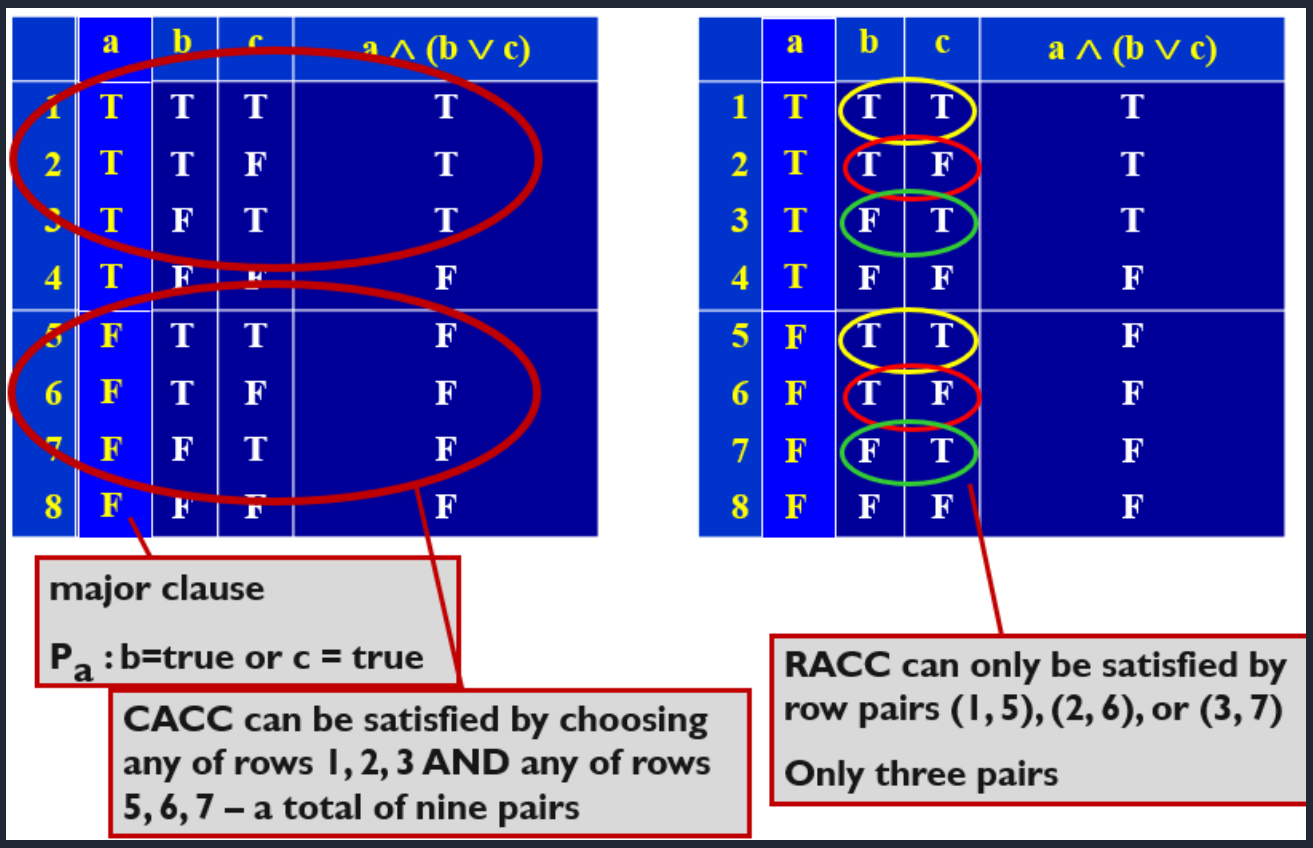
As we can see, PC does not fully exercise all the clauses. So, we can say that PC does not subsume CC.

3. b) Write down the characteristics of a Good Coverage Criterion.

Characteristics of a Good Coverage Criterion:

- Maximize the "bang for the buck", with fewer tests that are effective at finding more faults
- Provide traceability from software artifacts to tests
- Make regression testing easier
- Gives testers a stopping rule to finish the test
- Can be well supported with powerful tools.

3. c) For the expression, E=a AND (b OR c), determine CACC and RACC.



3. d) Consider the following graph:

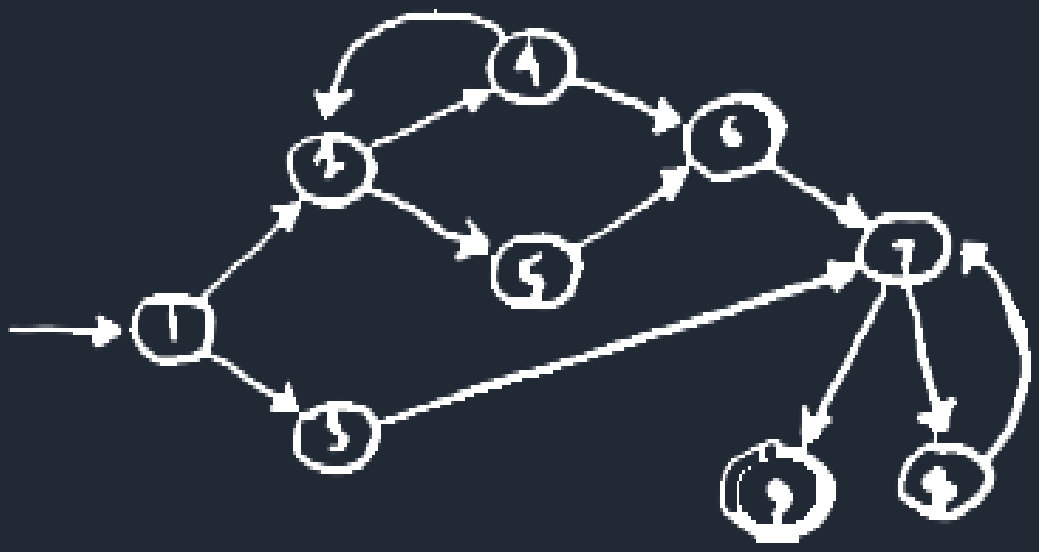


Figure-I

i. Set Representation of Figure-I graph.

N = {1,2,3,4,5,6,7,8,9}

N_0 = {1}

N_f = {9}

E = {(1,2), (1,3), (2,4), (2,5), (3,7), (4,2), (4,6), (5,6), (6,7), (7,8), (7,9), (8,7)}

ii. Prime Path Coverage Criteria of Figure-I graph.

PPC:

TR = {[2,4,2], [4,2,4], [4,2,5], [7,8,7], [8,7,8], [8,7,9], [1,3,7,8], [1,3,7,9], [1,2,4,6,7,9], [1,2,4,6,7,8], [1,2,5,6,7,9], [1,2,5,6,7,8]}

iii. Edge Coverage

TR= {(1,2), (1,3), (2,4), (2,5), (3,7), (4,2), (4,6), (5,6), (6,7), (7,8), (7,9), (8,7)}

TP: [1,3,7,8,7,9], [1,2,4,2,5,6,7,9], [1,2,4,6,7,9]

iv. Node Coverage

TR = {1,2,3,4,5,6,7,8,9}

TP: [1,2,4,6,7,8,7,9], [1,3,7,9], [1,2,5,6,7,9]

v. Edge-pair Coverage

TR= {[1,2,4], [1,2,5], [1,3,7], [2,4,2], [2,4,6], [2,5,6], [3,7,8], [3,7,9], [4,2,4], [4,6,7], [4,2,5] [5,6,7], [6,7,8], [6,7,9], [7,8,7], [8,7,8], [8,7,9]}

4. a) Define Input Domain. Write down the conditions for partitioning domains.

Input domain is the set of all possible inputs to a program.

Following are the conditions for partitioning domains:

- Blocks must be pairwise disjoint (no overlapping)

- Together, the blocks cover the domain D (complete)

4. b) Write down the advantages of input domain partitioning.

Following are the advantages of input domain partitioning:

- Can be equally applied at several levels of testing:
  - Unit Testing
  - Integration Testing
  - System Testing
- Relatively easy to apply with no automation
- Easy to adjust the procedure to get more or fewer tests
- No implementation is need, only needs the input space

4. c) See Page no. 7.

4. d) Use the following characteristics and blocks for the questions below.

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	+	-	x	/

i) Give test cases to satisfy the Each Choice Criterion.

Value 1	Value 2	Operation
-2	-2	+
0	0	-
2	2	*
2	2	/

ii) Give test cases to satisfy Base Choice Criterion. Assume base choices are Value 1 = > 0, Value 2 = >0, and Operation = +.

Value 1	Value 2	Operation
2	2	+
2	0	+
2	-2	+
0	2	+
-2	2	+
2	2	-
2	2	*
2	2	/

iii) How many test cases are needed to satisfy Pair-wise Coverage criterion?
4*3=12

Summer-2020

1. a) Define graph. Write down the set representation of the graph mentioned in Figure-I.

Any graph G(N, N_0, N_f, E), can be defined as:

- A non-empty set N of nodes
- A non-empty set N_0 of initial nodes
- A non-empty set N_f of initial nodes
- A set E of edges, each edge from one node to another (n_i, n_j), where i is the predecessor and j is the successor and $E \subseteq N \times N$.

See page-15.

1. b) Draw the control flow graph for each of the following code snippets.

```
x = 0;
while (x < y)
{
  y = f (x, y);
  if (y == 0)
  {
    break;
  } else if (y < 0)
  {
    y = y*2;
    continue;
  }
  x = x + 1;
}
```

(i)

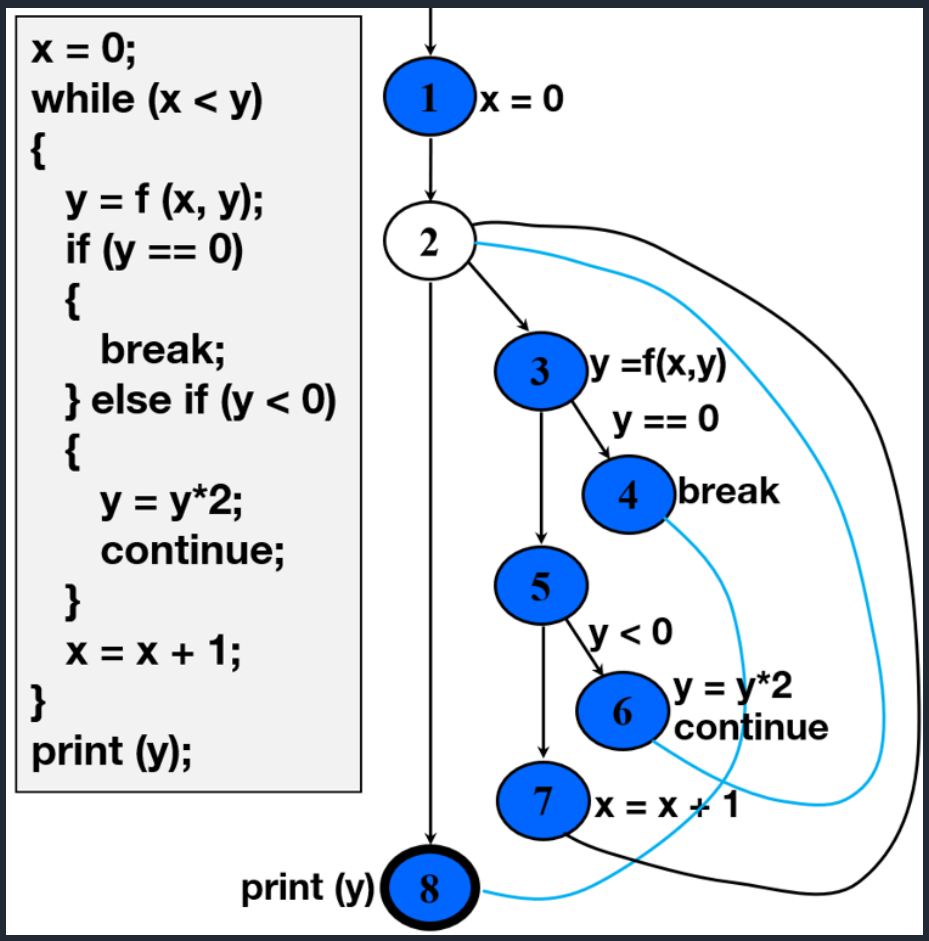
```
/**
   * Return index of node n at the
   * first position it appears,
   * -1 if it is not present
*/
public int indexOf (Node n)
{
 if(n!=null)
  {
     for (int i=0; i < path.size(); i++)
         if (path.get(i).equals(n))
             return i;
   }
     return -1;
}
```

(ii)

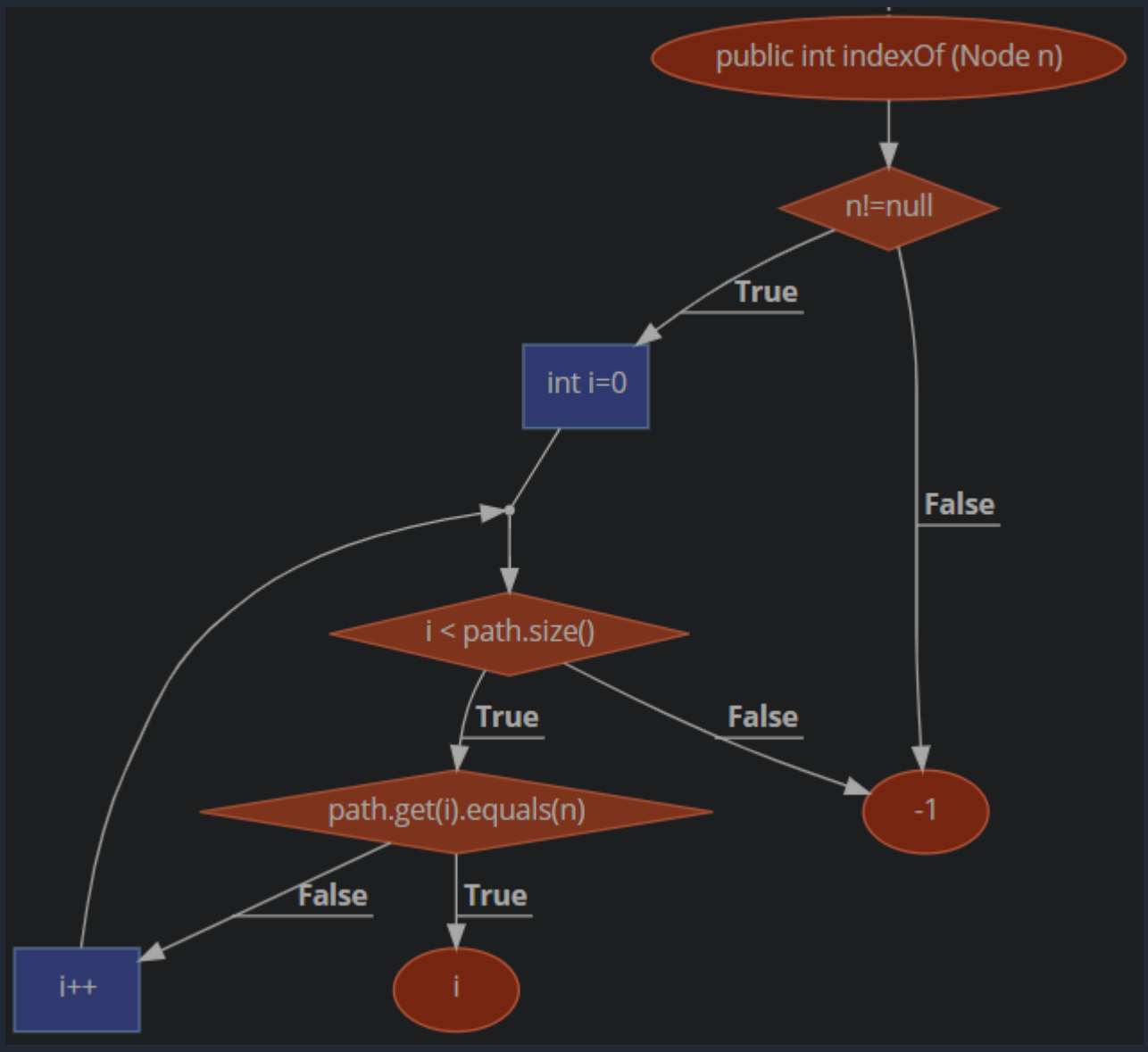
```
read ( c) ;
switch ( c )
{
  case 'N':
    z = 25;
  case 'Z':
    z += 20;
  case 'Y':
    x = 50;
    break;
  default:
    x = 0;
    break;
}
```

(iii)

For (i):

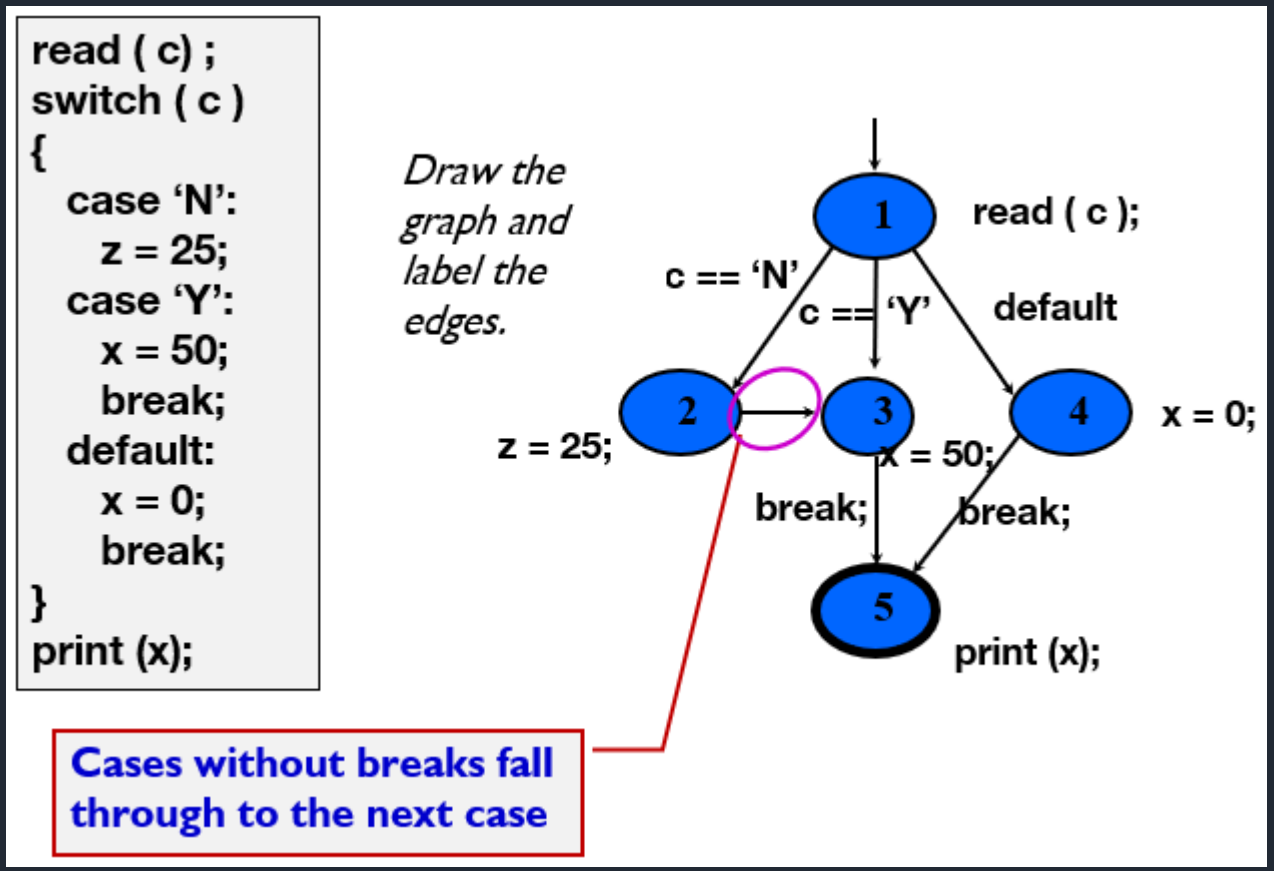

x = 0;
while (x < y)
{
y = f (x, y);
if (y == 0)
{
break;
} else if (y < 0)
{
y = y*2;
continue;
}
x = x + 1;
}
print (y);
1
x = 0
2
3
y =f(x,y)
y == 0
4
break
5
y < 0
6
y = y*2
continue
7
x = x + 1
print (y)
8

For (ii),


public int indexOf (Node n)
n!=null
True
False
int i=0
i < path.size()
True
False
path.get(i).equals(n)
-1
False
True
i++
i

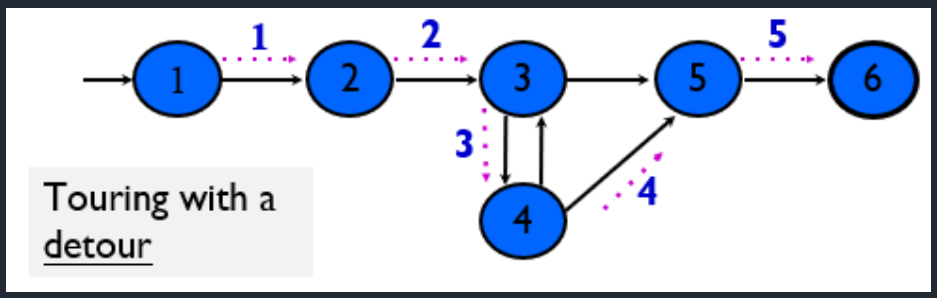
For (iii),

```
read ( c) ;
switch ( c )
{
   case 'N':
      z = 25;
   case 'Y':
      x = 50;
      break;
   default:
      x = 0;
      break;
}
print (x);
```

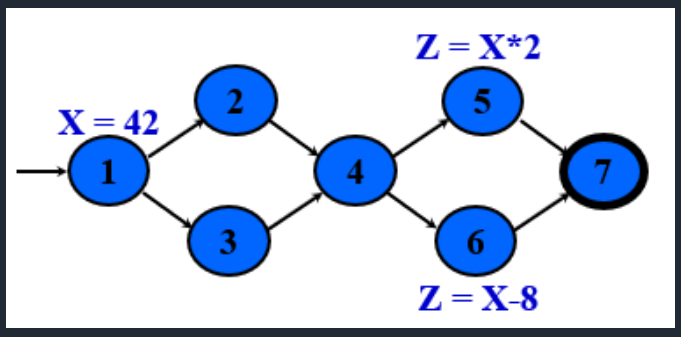


1. c) Define detour and def-clear path with appropriate example.

A test path p tours subpath q with detours if and only if every node in q is also in p in the same order.



A path from l_i to l_j is def-clear with respect to variable v, if v is not given another value on any of the nodes or edges in the path.



Here, a path [1,2,4,5] is def-clear with respect to x, as x was not given another value on any of the nodes or edges in the path.

1. d) Define def and use. How can we determine the defs of a particular variable from the source code?

Def is a location where a value for a variable is stored into memory.

Use is a location where a variable's value is accessed.

We can determine the defs of a particular variable from the source code by utilizing its corresponding control flow graph.

2. a) Define each of the followings:

Error: Software Error is an incorrect internal state that is the manifestation of some fault.

Failure: Software Failure is either an external or an internal behavior with respect to the requirements or other description of the expected behavior.

Mutation Testing: It is a type of white box testing in which, source code of one of the programs is changed and verifies whether the existing test cases can identify these defects in the system.

Happy Path Testing: It's a type of testing where the objective is to test an application successfully on a positive flow.

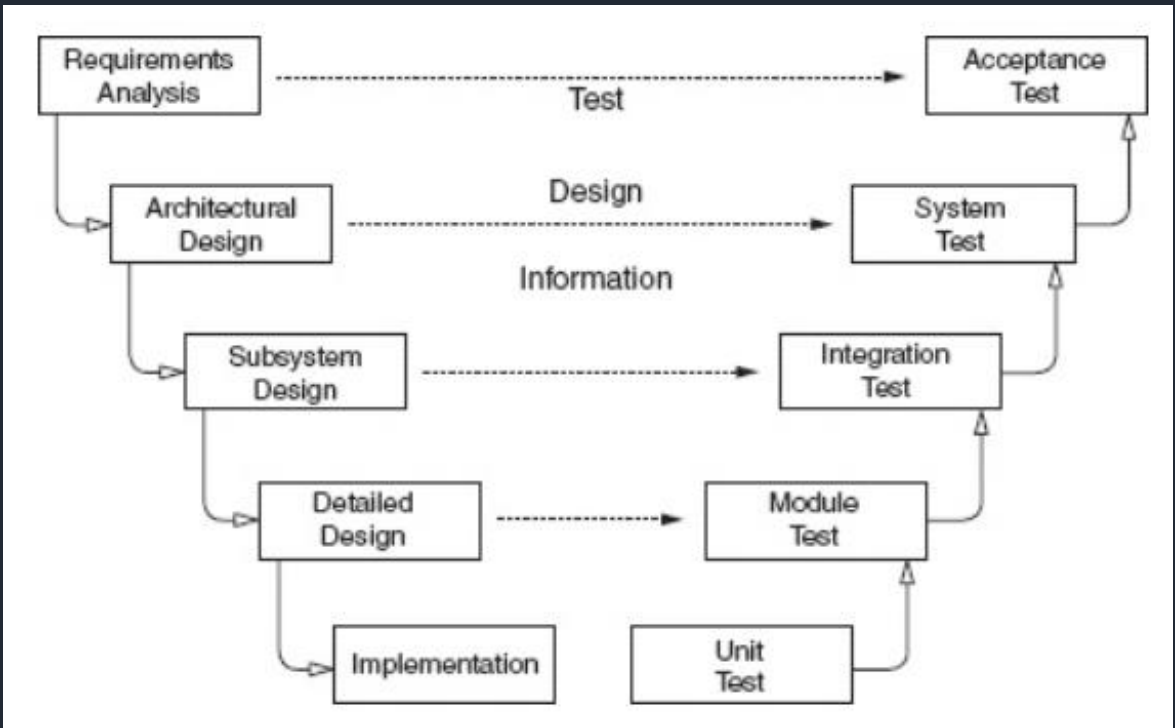
Stress Testing: This testing is done by putting the system under extremely heavy load beyond its capacity and specifications in order to check how and when it fails.

Regression Testing: Testing an application as a whole for the modifications in any module or functionality is termed as Regression Testing.

2. b) Give a comparison between functional and non-functional testing.

Functional Testing	Non-Functional Testing
Verifies each function or feature of the software.	Verifies non-functional aspects of the software such as performance, usability etc.
Can be done manually.	Is difficult to perform manually.
It is based on customer's requirements.	It is based on customer's expectation.
Its goal is to validate software actions.	Its goal is to validate the performance of the software.
It describes what a product does.	It describes how a product works.
It is performed before non-functional testing.	It is performed after functional testing.
An example is to check the login functionality of a software.	An example is to check if the dashboard loads in 2 seconds.

2. c) With necessary diagram, briefly describe the software testing V-model.



Requirement analysis phase captures customers' needs.

Acceptance testing is designed to determine whether the completed software in fact meets those needs. It must involve users or other individuals who have strong domain knowledge.

Architectural design phase chooses components and connectors that together realizes a system whose specification is intended to meet the previously identified requirements.

System testing determines whether the assembled system meets its specifications. It assumes that each piece work individually and asks if the system works as a whole.

Subsystem design phase specifies the structure and behavior of subsystems.

Integration testing is designed to assess whether the interfaces between modules in a subsystem have consistent assumptions and communicate correctly.

Detailed design phase determines structure and behavior of individual modules.

Module testing is designed to assess individual modules in isolation, including how the component units interact with each other and their associated data structures. It is also called developer testing since most software development companies use their programmers to conduct module testing.

Implementation phase produces the code.

Unit testing is designed to assess the units produced by the implementation phase. It's the lowest level of testing. Similar to module testing, it is also called developer testing.

3. a) Define Input Domain. Write down the properties of domain partitioning.

Input domain is the set of all possible inputs to a program.

Following are the properties of domain partitioning:

- If the partitions are not complete or disjoint, that means the partitions have not been considered carefully enough
- They should be reviewed carefully, like any design
- Different alternatives should be considered
- We model the input domain in five steps:
    - Steps 1 and 2 move us from the implementation abstraction level to the design abstraction level
    - Steps 3 & 4 are entirely at the design abstraction level
    - Step 5 brings us back down to the implementation abstraction level

3. b) Define Coverage Criteria. Write down the advantages of Input Space Partitioning.

Coverage criteria are usually defined as rules or requirements, which a test suite needs to satisfy. It provides structured, practical ways to search the input space.

Advantages of ISP:

- Can be equally applied at several levels of testing
    - Unit
    - Integration
    - System
- Relatively easy to apply with no automation
- Easy to adjust the procedure to get more or fewer tests
- No implementation knowledge is needed
    - Just the input space

3. c) See page 16.

3. d) Briefly describe the five steps of input domain modeling.

Step 1: Identify testable functions

- Individual methods have one testable function
- Methods in a class often have the same characteristics
- Programs have more complicated characteristics—modeling documents such as UML can be used to design characteristics
- Systems of integrated hardware and software components can use devices, operating systems, hardware platforms, browsers, etc.

Step 2: Find all the parameters

- Often fairly straightforward, even mechanical
- Important to be complete

- Methods: Parameters and state (non-local) variables used
- Components: Parameters to methods and state variables
- System: All inputs, including files and databases

Step 3: Model the input domain

- The domain is scoped by the parameters
- The structure is defined in terms of characteristics
- Each characteristic is partitioned into sets of blocks
- Each block represents a set of values
- This is the most creative design step in using ISP

Step 4: Apply a test criterion to choose combinations of values

- A test input has a value for each parameter
- One block for each characteristic
- Choosing all combinations is usually infeasible
- Coverage criteria allow subsets to be chosen

Step 5: Refine combinations of blocks into test inputs

- Choose appropriate values from each block

4. a) Define Logic Coverage. Write down the sources of logic expression and predicates.

Logic corresponds to the internal structure of the code and this testing is adopted for safety-critical applications. This verifies the subset of the total number of truth assignments to the expressions.

See 3. a) of Page 5.

4. b) Define determination in logic coverage. For the logic expression, p = (x AND y) OR (x AND (NOT y)), show that only clause x determines the predicate.

$$\underline{p = (a \wedge b) \vee (a \wedge \neg b)}$$

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} \\ &= ((true \wedge b) \vee (true \wedge \neg b)) \oplus ((false \wedge b) \vee (false \wedge \neg b)) \\ &= (b \vee \neg b) \oplus false \\ &= true \oplus false \\ &= true \end{aligned}$$

$$p = (a \wedge b) \vee (a \wedge \neg b)$$

$$\begin{aligned} p_b &= p_{b=true} \oplus p_{b=false} \\ &= ((a \wedge true) \vee (a \wedge \neg true)) \oplus ((a \wedge false) \vee (a \wedge \neg false)) \\ &= (a \vee false) \oplus (false \vee a) \\ &= a \oplus a \\ &= false \end{aligned}$$

- *a* always determines the value of this predicate
- *b* never determines the value – *b* is irrelevant !

4. c) CACC vs RACC.

CACC	RACC
Only allows major clause to have different values.	Implicitly allows minor clauses to have different values.
The values chosen for the minor clauses must be the same	The values chosen for the minor clauses must cause predicate to be true for one value, and false for the other
CACC subsumes PC	RACC does not subsume such coverages
Common interpretation	Recent interpretation

4. d) See Page 15.