Internal Assessment Test II – April 2018

Sub:	Software Testing					Sub Code:	10CS842	Branch:	CSE
Date:	17/04/ 2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VIII A,B,C		OBE

1. Explain in detail about mc cabe's basis path method using graph theory. (10M)

McCabe based his view of testing on a major result from graph theory, which states that the cyclomatic number for a strongly connected graph is the number of linearly independent circuits in the graph.
McCabe developed an algorithmic procedure called as baseline method to determine a set of basic paths.

BASIS PATH TESTING EXAMPLE

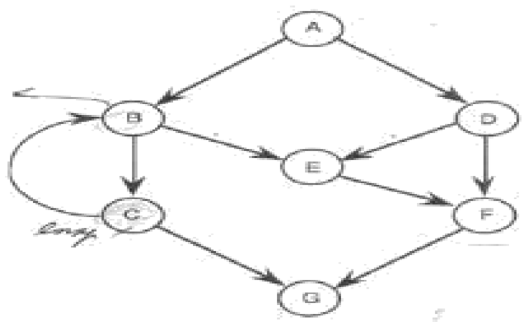


Figure 9.6 McCabe's Control Graph

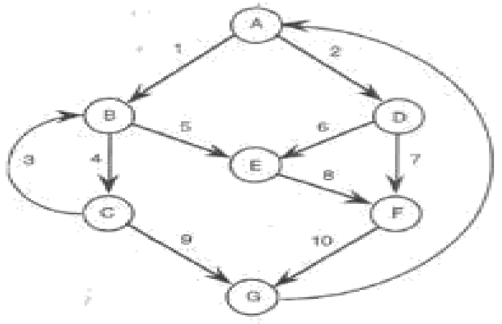


Figure 9.7 McCabe's Derived Strongly Connected Graph

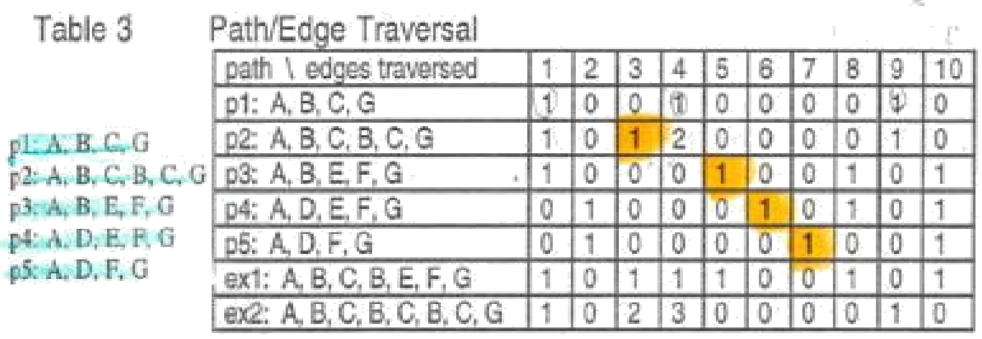


Table 3 Path/Edge Traversal

path \ edges traversed	1	2	3	4	5	6	7	8	9	10
p1: A, B, C, G	1	0	0	1	0	0	0	0	1	0
p2: A, B, C, B, C, G	1	0	1	2	0	0	0	0	1	0
p3: A, B, E, F, G	1	0	0	0	1	0	0	1	0	1
p4: A, D, E, F, G	0	1	0	0	0	1	0	1	0	1
p5: A, D, F, G	0	1	0	0	0	0	1	0	0	1
ex1: A, B, C, B, E, F, G	1	0	1	1	1	0	0	1	0	1
ex2: A, B, C, B, C, B, C, G	1	0	2	3	0	0	0	0	1	0

p1: A, B, C, G
p2: A, B, C, B, C, G
p3: A, B, E, F, G
p4: A, D, E, F, G
p5: A, D, F, G

2. Explain slice based testing in data flow testing.

Slice Based Testing: (1M)
A program slice is a set of program statements that contribute to, or affect a value for a variable at some point in the program. The idea of slicing is to divide a program into components that have some useful meaning.
The idea of slices is to separate a program into components that have some useful (functional meaning).

Slice Based Testing:

Definition: **(1M)**
Given a program P, program graph G(P) and a set V of variables in P, a slice on the variable set V at statement n, written S(V, n), is the set of all statements in P prior to node n that contribute to the values of variables in V at node n.

- "prior to": a slice captures the execution time behavior of a program with respect to the variable(s) in the slice
- "contribute": some declarative statements have an effect on the value of a variable (e.g. const, type)

The USE relationship pertains to five forms of usage:

– P-use: used in a predicate (decision)
– C-use: used in a computation
– O-use: used for output
– L-use: used for location (e.g. pointers)
– I-use: used for iteration (internal counters, loop indices)

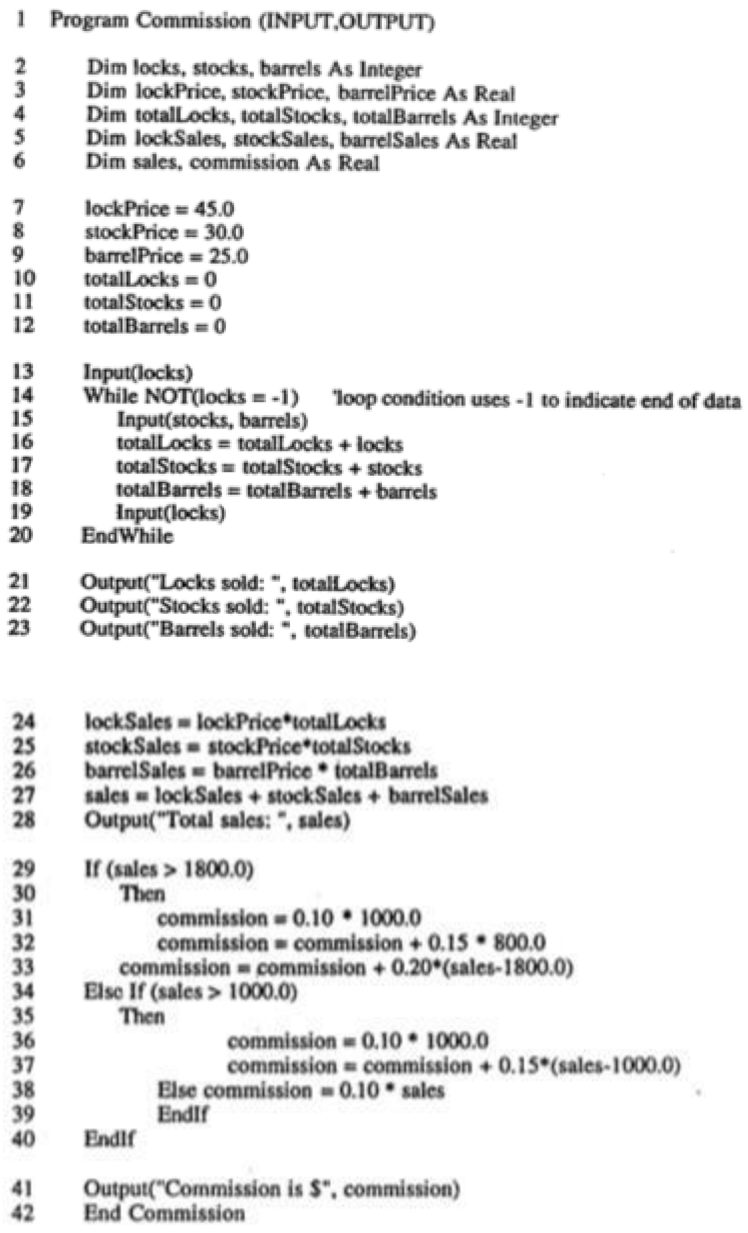
Definition nodes:
– I-def: defined by input
– A-def: defined by assignment
Eventually we will develop a DAG (directed acyclic graph) of slices in which nodes are slices and edges correspond to the subset relationship.

Guidelines for choosing slices:
– If the value of v is the same whether the statement fragment is included or excluded, exclude the fragment.
– If statement fragment n is:

- a defining node for v, include n in the slice
- a usage node for v, exclude n from the slice

Note: – O-use, L-use & I-use nodes are excluded from slices

(3M)



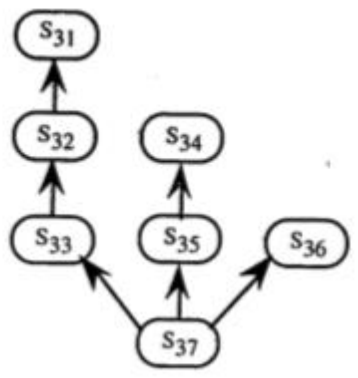
```
1  Program Commission (INPUT,OUTPUT)

2      Dim locks, stocks, barrels As Integer
3      Dim lockPrice, stockPrice, barrelPrice As Real
4      Dim totalLocks, totalStocks, totalBarrels As Integer
5      Dim lockSales, stockSales, barrelSales As Real
6      Dim sales, commission As Real

7      lockPrice = 45.0
8      stockPrice = 30.0
9      barrelPrice = 25.0
10     totalLocks = 0
11     totalStocks = 0
12     totalBarrels = 0

13     Input(locks)
14     While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15         Input(stocks, barrels)
16         totalLocks = totalLocks + locks
17         totalStocks = totalStocks + stocks
18         totalBarrels = totalBarrels + barrels
19         Input(locks)
20     EndWhile

21     Output("Locks sold: ", totalLocks)
22     Output("Stocks sold: ", totalStocks)
23     Output("Barrels sold: ", totalBarrels)


24     lockSales = lockPrice*totalLocks
25     stockSales = stockPrice*totalStocks
26     barrelSales = barrelPrice * totalBarrels
27     sales = lockSales + stockSales + barrelSales
28     Output("Total sales: ", sales)

29     If (sales > 1800.0)
30         Then
31             commission = 0.10 * 1000.0
32             commission = commission + 0.15 * 800.0
33         commission = commission + 0.20*(sales-1800.0)
34     Else If (sales > 1000.0)
35         Then
36                 commission = 0.10 * 1000.0
37                 commission = commission + 0.15*(sales-1000.0)
38             Else commission = 0.10 * sales
39             EndIf
40     EndIf

41     Output("Commission is $", commission)
42     End Commission
```

As per commission problem we can identify slices. (3M)

- S1: S(locks,13) = {13} (defining node I-def)
- S2: S(locks,14) = {13,14,19,20}
- S3: S(locks,16) = {13,14,16,19,20}
- S4: S(locks,19) = {19} (defining node I-def)
- S5: S(stocks,15) = {13,14,15,19,20}
- S6: S(stocks,17) = {13,14,15,17,19,20}
- S7: S(barrels,15) = {13,14,15,19,20}
- S8: S(barrels,18) = {13,14,15,18,19,20}
- S9: S(totallocks,10) = {10} (A-def)

• S10: S(totallocks,16) = {10,13,14,16,19,20}
(A-def & C-use) S(totallocks,21)= {10,13,14,16,19,20} 21 is an O-Use of totallocks, excluded
• S11: S(totallocks,24) = {10,13,14,16,19,20} (24 is a C-use of total locks)
• S12: S(totalstocks,11) = {11} (A-def)
• S13: S(totalstocks,17) = {11,13,14,15,17,19,20} (A-def & C-use)
• S14: S(totalstocks,22) = {11,13,14,15,17,19,20}(22 is an O-Use of totalstocks)
• S15: S(totalbarrels ,12) = {12}
• S16: S(totalbarrels,18) = {12,13,14,15,18,19,20} (A-def & C-use)
• S17: S(totalbarrels,23) = {12,13,14,15,18,19,20} (23 is an O-Use of totalbarrels)
• S18: S(lock_price,24) = {7} (A-def)
• S19: S(stock_price,25) = {8}
• S20: S(barrel_price,26) = {9}
• S21: S(lock_sales,24) = {7,10,13,14,16,19,20,24}
• S22: S(Stock_sales,25) = {8,11,13,14,15,17,19,20,25}
• S23: S(barrel_sales,26) = {9,12,13,14,15,18,19,20,26}

Slice information improves our insight. We can build lattice which is a directed acyclic graph in which slices are nodes and an edge represents the proper subset relationship as show in figure. (2M)



Lattice of slices on commission.

3. Define the program graph . Write structures triangle program and the program graph.

Program Graph: G(P) (2M)
Statement fragments (or entire statements) are nodes, Edges represent node sequence G(P) has a single entry node & single exit node and there are no edges from a node to itself.
• Node *n* in the CFG of program P is a **defining node** of the variable *v* V, written as DEF(*v*, *n*), iff the value of the variable *v* is unambiguously defined at the statement fragment corresponding to node *n*

• Node *n* in the CFG of program P is a **usage node** of the variable *v* V, written as USE(*v*, *n*), iff the value of the variable *v* is used at the statement fragment corresponding to node *n*.

• A usage node USE(*v*, *n*), is a predicate use (denoted as P-use) iff the statement n is a predicate statement, otherwise USE(*v*, *n*) is a computation use or C-use. – The nodes corresponding to predicate uses have always an outdegree ≥ 2, and nodes corresponding to computation uses always have outdegree ≤ 1

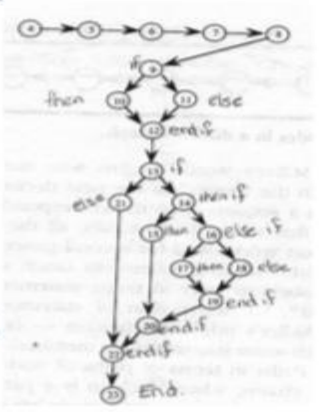
A **definition-use** (sub) path with respect to a variable *v* (denoted as du-path) is a (sub)path in PATHS(P), where PATHS(*P*) is the set of all possible paths in the CFG of program *P*, such that, for some *v* in *V*, there are define and usage nodes DEF(*v*, *m*) and USE(*v*, *n*) such that *m*, and *n* are the initial and final nodes in the path respectively.

A **definition-clear** (sub) path with respect to a variable *v* (denoted as dc-path) is a definition-use path in PATHS(*P*) with initial nodes DEF(*v*, *m*) and USE(*v*, *n*) such that there no other node in the path is a defining node for *v*

(4M)

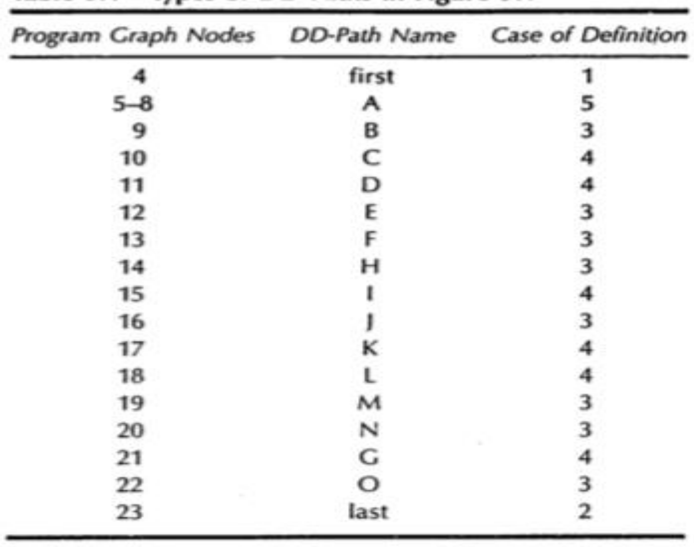
A program graph for the Triangle Problem

```
1.  Program triangle (input, output);
2.  VAR a,b,c          : integer;
3.        IsATriangle  : boolean;
4.  writeln ('Enter 3 integers which are sides of a triangle:');
5.  readln (a,b,c);
6.  writeln ('Side A is ', a);
7.  writeln ('Side B is ' ,b);
8.  writeln ('Side C is ' ,c);
9.  IF  ( a < b+c ) AND ( b < a+c ) AND ( c < a+b )
10.      THEN isATriangle := True;
11.      ELSE  isATriangle := False;
12. ENDIF;
13. IF isATriangle
14.     THEN  IF  ( a=b ) AND ( b=c )
15.             THEN writeln ('Triangle is Equilateral');
16.             ELSE IF ( a≠b ) AND ( a≠c ) AND ( b≠c )
17.                     THEN writeln ('Triangle is Scalene');
18.                     ELSE writeln ('Triangle is Isosceles');
19.                     ENDIF;
20.             ENDIF;
21. ELSE writeln ('Not a Triangle');
22. ENDIF;
23. END.
```



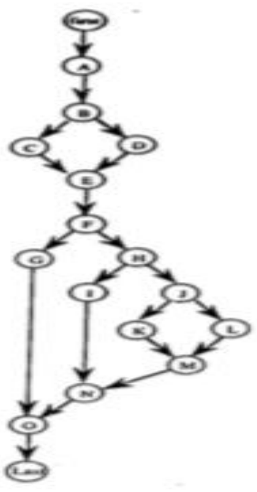
(2M)

Table 9.1 Types of DD-Paths in Figure 9.1



Program Graph Nodes	DD-Path Name	Case of Definition
4	first	1
5–8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	last	2

(2M)



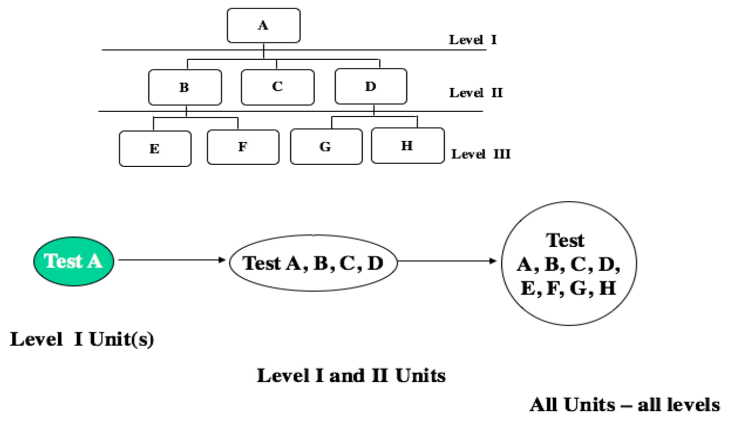
DD-Path graph for the triangle program.

4. Explain the top down and bottom up integration strategies.

(5M)

Top-Down Integration Testing Strategy

Top-down Integration Testing Strategy



Top-down integration strategy focuses on testing the top layer or the controlling subsystem first (i.e. the *main*, or the root of the call tree).

- The general process in top-down integration strategy is to gradually add more subsystems that are referenced/required by the already tested subsystems when testing the application
- Do this until all subsystems are incorporated into the test
- Special program is needed to do the testing, *Test stub:*
- A program or a method that simulates the input-output functionality of a missing subsystem by answering to the decomposition sequence of the calling subsystem and returning back simulated or "canned" data.

Advantages and Disadvantages of Top-Down Integration Testing

• Test cases can be defined in terms of the functionality of the system (functional requirements) and Structural techniques can also be used for the units in the top levels.
• Writing stubs can be difficult especially when parameter passing is complex. Stubs must allow all possible conditions to be tested.
• Possibly a very large number of stubs may be required, especially if the lowest level of the system contains many functional units.
• One solution to avoid too many stubs: *Modified top-down testing strategy (Bruege)*
Test each layer of the system decomposition individually before merging the layers.

(5M)
Bottom-Up Integration Testing Strategy

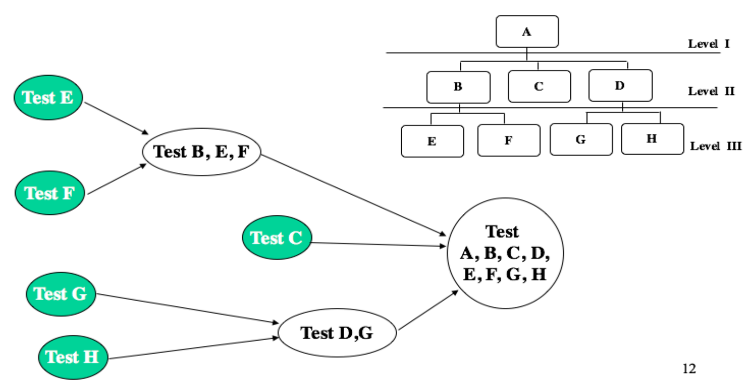
• Bottom-Up integration strategy focuses on testing the units at the lowest levels first (i.e. the units at the leafs of the decomposition tree).

• The general process in bottom-up integration strategy is to gradually include the subsystems that reference/require the previously tested subsystems

• This is done repeatedly until all subsystems are included in the testing

• Special program called *Test Driver* is needed to do the testing.
– The Test Driver is a "fake" routine that requires a subsystem and passes a test case to it.

Example Bottom-Up Strategy



Advantages and Disadvantages of Bottom-Up Integration Testing

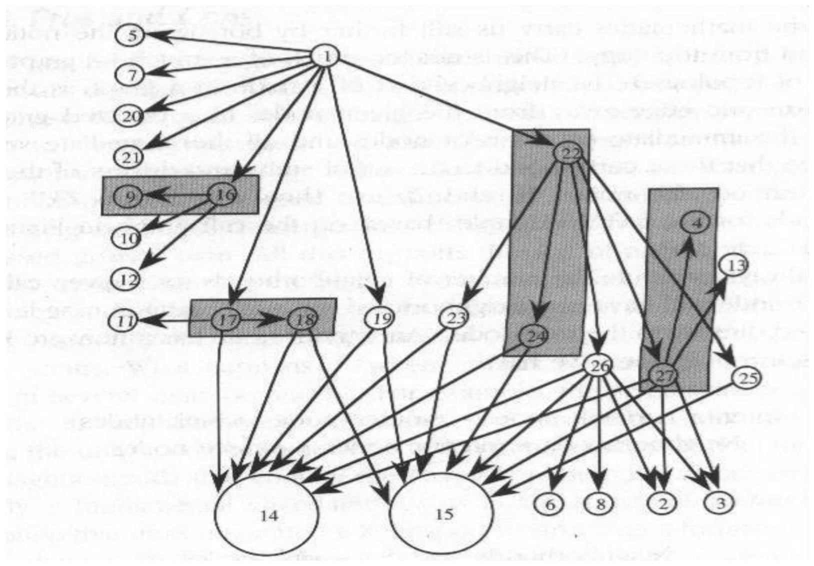
Not optimal strategy for functionally decomposed systems.
– Tests the most important subsystem (UI) last.
• Useful for integrating the following systems
– Object-oriented systems.
– Real-time systems.
– Systems with strict performance requirements.

5. Explain call graph based integration with the help of
1) Pair wise integration and (5M)
2) Neighborhood integration.(5M)

The idea behind Pair-Wise integration testing is to eliminate the need for developing stubs/drivers

- The objective is to use actual code instead of stubs/drivers.
- In order to deteriorate the process to a big-bang strategy, we restrict a testing session to just a pair of units in the call graph
- The result is that we have one integration test session for each edge in the call graph
- Pair wise integration takes 40 sessions in SATM vs. 42 of top-down or bottom-up.

There is not much reduction in sessions from top-down or bottom-up, but it is a drastic reduction in stub/driver development effort.



Neighborhood Integration

• We define the neighborhood of a node in a graph to be the set of nodes that are one edge away from the given node.
• In a directed graph means all the immediate predecessor nodes and all the immediate successor nodes of a given node.
• The number of neighborhoods for a given graph can be computed as: InteriorNodes = nodes – (SourceNodes + SinkNodes)Neighborhoods = InteriorNodes + SourceNodes
Or Neighborhoods = nodes – SinkNodes
• Neighborhood Integration Testing reduces the number of test sessions
• For a given call graph, there is one neighborhood for each interior node, plus one extra in case there are leaf nodes connected directly to the root node

Node	**Predecessors**	**Successors**
16	1	9, 10, 12
17	1	11, 14, 18
18	17	14, 15
19	1	14, 15
23	22	14, 15
24	22	14, 15
26	22	14, 15, 6, 8,2,3 2, 3
27	22	14, 15, 2, 3, 4, 13
25	22	15
22	1	23, 24, 26, 27, 25
1	n/a	5, 7, 2, 21, 16, 17, 19, 22

Neighborhood integration yields a drastic reduction in the number of integration test sessions without stubs & drivers.

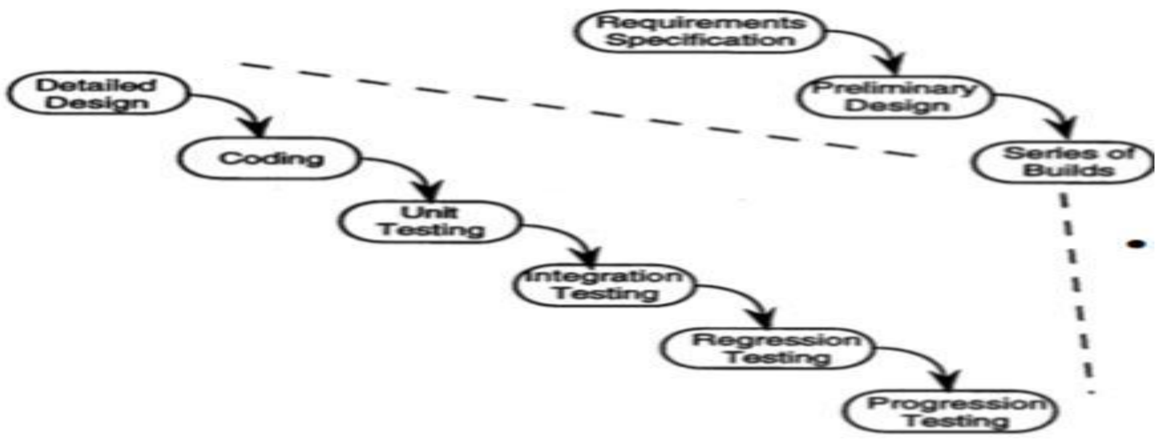
Advantages of call graph based integration:

• Call graph based integration techniques move towards a behavioral basis.
• Aim to eliminate / reduce the need for drivers/stubs.
• Closer to a build sequence.
• Neighborhoods can be combined to create "villages".
• Suffer from the fault isolation problem especially for large neighborhoods. What happens if a fault is found in a node (unit) that appears in several neighborhoods?
Example: The screen driver unit appears in 7 of the 11 neighborhoods. When we resolve the fault, we change the unit's code, and thus we need to retest all the previously tested neighborhoods that contain the changed node

6. With neat diagram explain the traditional view of testing levels of waterfall life cycle (5M) rapid prototyping life cycle.(5M)

The normal waterfall phases from detailed design through integration testing occur. System testing is split into;

- Regression testing
- Progression testing



7. Define the below terms: i) Threads ii) MM-Paths. iii) Data. iv) Action v) Ports

Action (2M)
Action-centered modeling is a common form for requirements specification

- Actions have input and output (Either data or port events)

- Synonyms: Transform, data transform, control transform, process, activity, task, method and service.
- Used in functional testing
- They can be refined (decomposed)
- It's the Basis of structural testing

Threads (2M)
A system thread is a path from a source ASF to a sink ASF in the ASF graph of a system.ASF (Atomic System Function): is an action that is observable at the system level in terms of port input and output events.ASF Graph: of the system is the directed graph in which nodes are ASFs and edges represent sequential flow. Or Atomic system function graph – ASF graph

- A directed graph where Nodes are ASFs, Edges represent sequential flow.
- Source / Sink atomic system function: A source / sink node in an ASF

System thread: A path from a source ASF to a sink ASF Thread graph: A directed graph where Nodes are system threads, Edges represent sequential execution of threads.

Devices (Ports) (2M)
Port input and output handled by devices

- A port is a point at with an I/O device is attached to a system
- Physical actions occur on devices and enter / leave system through ports
- Physical to logical translation on input
- Logical to physical translation on output
- System testing can be moved to the logical level – ports (No need for devices)
- Thinking about ports helps testers define input space and output space for functional testing.

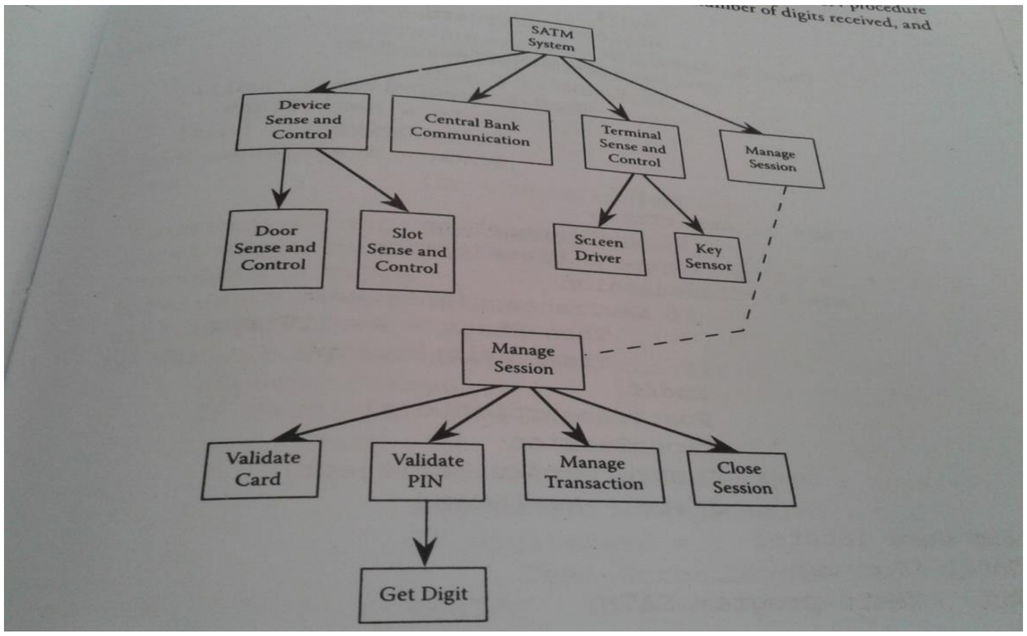
MM-Paths (Module Message paths) (2M)
A source node in a program is a statement fragment at which program execution begins or resumes.A sink node in a unit is a statement fragment at which program execution terminates.A module execution path is a sequence of statements that begins with a source node and ends with a sinknode, with no intervening sink nodes.A message is a programming language mechanism by which one unit transfers control to another unit.
A MM-Path is an interleaved sequence of module execution paths and messages.

Data: (2M)

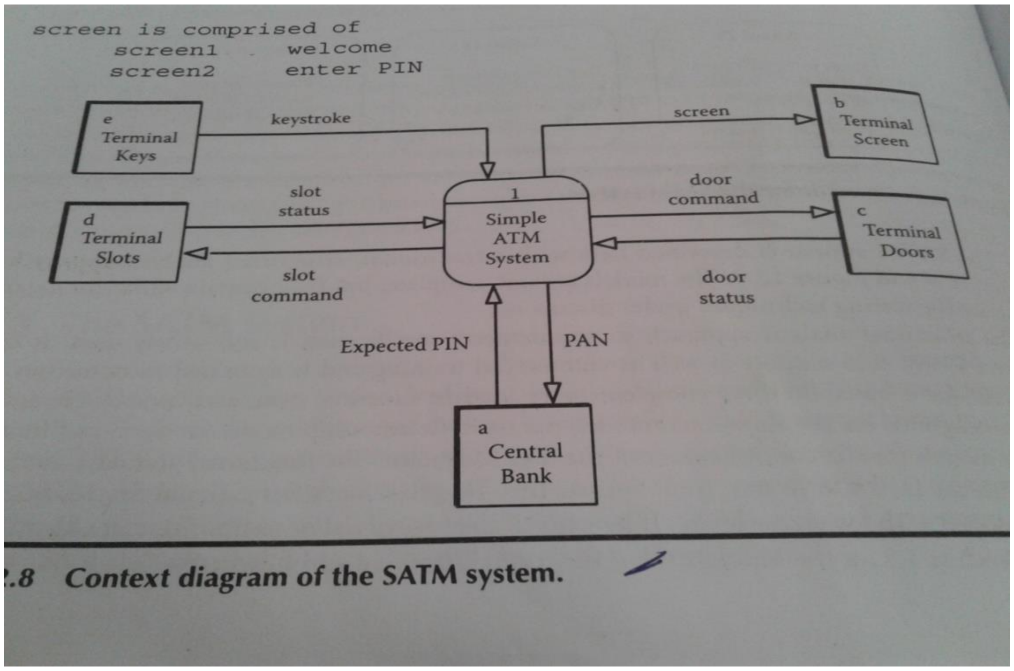
- Focus on information used and created by the system
- Data is described using Variables, data structures, fields, records, data stores and files Entity-relationship models describe highest level & Regular expressions used at more detailed level.
- Data view: Good for transaction view of systems

8. Briefly explain about SATM system, draw the context diagram , ER model and decomposition tree for SATM system.

SATM- 1M
Context diagram- 3 M
ER diagram – 3 M
Decomposition tree -3 M

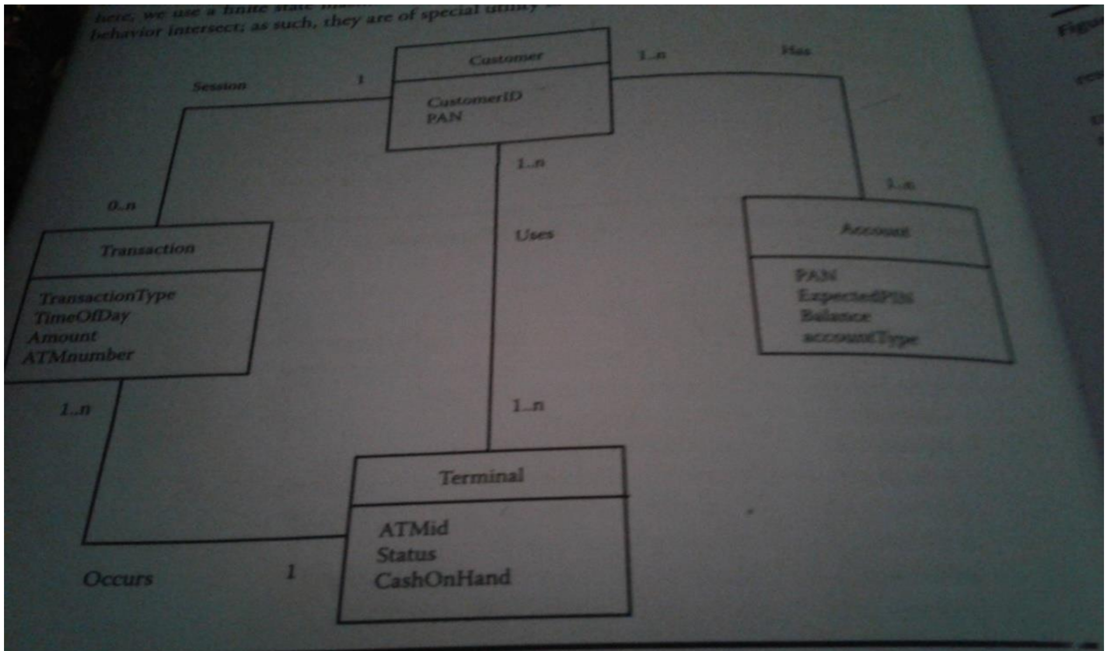


Decomposition tree of SATM



Context Diagram of SATM

here, we use a finite state
behavior intersect; as such, they are of special utility



10 *Entity/relationship model of the SATM system.*

ER model of SATM