CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 2 – April. 2018

Scheme and Solution

Sub:	Software Testing							Code:	15IS63
Date :	17/ 04/2018	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	ISE

Note: Answer any five questions:

1. a) Explain the basic decision table terms (6 M)

A Decision Table is the method used to build a complete set of test cases without using the internal structure of the program in question. In order to create test cases we use a table to contain the input and output values of a program. Sucha table is split up into four sections as shown below in fig

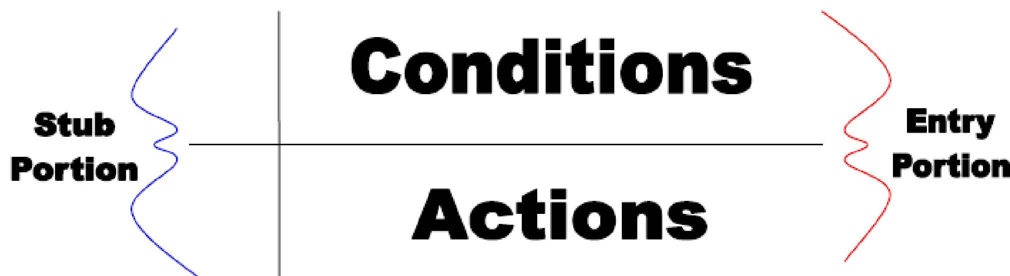


Figure 2.1 The Basic Structure of a Decision Table.

In fig there are two lines which divide the table into its main structure. The solid vertical line separates the Stub and Entry portions of the table, and the solid horizontal line is the boundary between the Conditions and Actions. So these lines separate the table into four portions, Condition Stub, Action Stub, Condition Entries and Action Entries.

A column in the entry portion of the table is known as a *rule*. Values which are in the condition entry columns are known as inputs and values inside the action entry portions are known as outputs. Outputs are calculated depending on the inputs and specification of the program.

In fig below there is an example of a typical Decision Table. The inputs in this given table derive the outputs depending on what conditions these inputs meet. Notice the use of "-"in the table below, these are known as *don't care entries*. Don't care entries are normally viewed as being false values which don't require the value to define the output.

Stub	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
c1	T	T	T	F	-	T
c2	F	T	T	T	-	T
c3	T	T	-	T	T	T
c4	T	F	F	T	T	T
a1	X	X		X	X	X
a2		X				
a3	X			X		
a4			X			X

Figure 2.2 shows its values from the inputs as true(T) or false(F) values which are binary conditions, tables which use binary conditions are known as *limited entry decision tables*. Tables which use multiple conditions are known as *extended entry decision tables*. One important aspect to notice about decision tables is that they aren't imperative as that they don't apply any particular order over the conditions or actions.

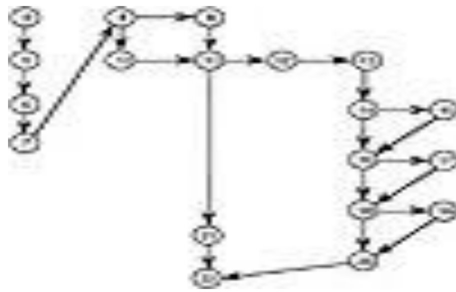
b) Define regression and progression testing. (4 M)

The goal of regression testing is to assure that things that worked correctly in the previous build still work with the newly added code. Progression testing assumes that regression testing was successful, and that the new functionality can be tested.

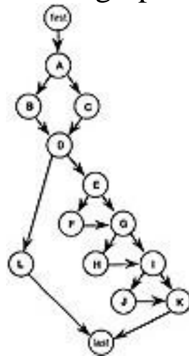
Regression testing is an absolute necessity in a series of builds because of the well-known "ripple effect" of changes to an existing system. (The industrial average is that one change in five introduces a new fault.)

2. Explain in detail Basis Path Testing with respect to triangle problem (10 M)

```
1.  program triangle (input, output) ;
2.  VAR a, b, c  : integer;
3.    IsATriangle     : boolean;
4.  BEGIN
5.  writeln('Enter three integers which are sides of a triangle:');
6.  readln (a,b,c);
7.  writeln('Side A is ',a, 'Side B is ',b, 'side C is ',c);
8.  IF (a < b + c) AND (b < a + c) AND (c < a + b)
9.    THEN IsATriangle :=TRUE
10.   ELSE IsATriangle := FALSE ;
11. IF IsATriangle
12.   THEN
13.   BEGIN
14.   IF (a = b) XOR (a = c) XOR (b = c) AND NOT((a=b) AND (a=c))
15.     THEN Writeln ('Triangle is Isosceles') ;
16.   IF (a = b) AND (b = c)
17.     THEN Writeln ('Triangle is Equilateral') ;
18.   IF (a <> b) AND (a <> c) AND (b <> c)
19.     THEN Writeln ('Triangle is Scalene') ;
20.   END
21.   ELSE WRITELN('Not a Triangle') ;
22. END.
```



DD-Path graph of a program.



p1: A-B-D-E-G-I-J-K-Last
p2: A-C-D-E-G-I-J-K-Last
p3: A-B-D-L-Last
p4: A-B-D-E-F-G-I-J-K-Last
p5: A-B-D-E-F-G-H-I-J-K-Last
p6: A-B-D-E-F-G-H-I-K-Last

if you follow paths p2, p3, p4, p5, and p6, you find that they are all infeasible. Path p2 is infeasible, because passing through node C means the sides are not a triangle, so none of the sequel decisions can be taken. Similarly, in p3, passing through node B means the sides do form a triangle, so node L cannot be traversed. The others are all infeasible because they involve cases where a triangle is of two types (e.g., isosceles and equilateral). The problem here is that there are several inherent dependencies in the triangle problem. One is that if three integers constitute sides of a triangle, they must be one of the three possibilities: equilateral, isosceles, or scalene. A second dependency is that the three possibilities are mutually exclusive: if one is true, the other two must be false.

fp1: A-C-D-L-Last	(Not a triangle)
fp2: A-B-D-E-F-G-I-K-Last	(Isosceles)
fp3: A-B-D-E-G-H-I-K-Last	(Equilateral)

fp4: A-B-D-E-G-I-J-K-Last	(Scalene)

3. Briefly explain test coverage metrics (5 M)

C_0: Every statement

C_1 :Every DD-Path (predicate outcome)

C_{1p}: Every predicate to each outcome

C_2: C_1 coverage + loop coverage

C_d: C_1 coverage + every dependent pair of DD-Paths

C_{MCC}:Multiple condition coverage

C_{ik} :Every program path that contains up to k repetitions of a loop (usually k = 2)

C_{stat}: Statistically significant" fraction of paths

C_∞:All possible execution paths

Explain the terms: oracle, scaffolding, and self check oracles(5 M)

If a software test is a sequence of activities (*stimuli and observations*), an **oracle** is a predicate that determines whether a given sequence is acceptable or not.
An oracle will respond with a *pass* or a *fail* verdict on the acceptability of any test sequence for which it is defined.
A test oracle is **complete** if it can offer a verdict for any set of test input.
● A test oracle is **sound** if it offers the right verdict for any test case that it can offer a verdict for.
● A test oracle is **correct** if it is both sound and complete.
○ It is *partially correct* if it is sound, but not complete.

Self-Check oracle
Usually written at the function level.
○ For one method or one high-level "feature".
○ Properties based on behavior of that function.
● Work for any input to that function.
● Only accurate for those properties.
○ Faults may be missed if the specified properties are obeyed.
○ More properties = more expensive to write.

4. Explain the SATM application with the help of (i) Level 1 Data flow diagram ii) Upper level finite state machine.

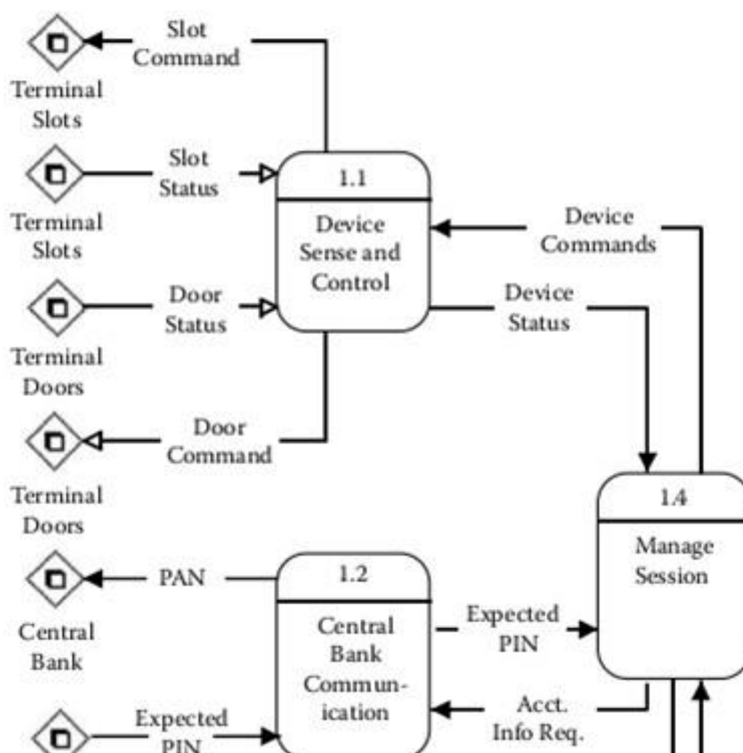
Level 1 Data flow diagram(5 M)

The structured analysis approach to requirements specification is the most widely used method in

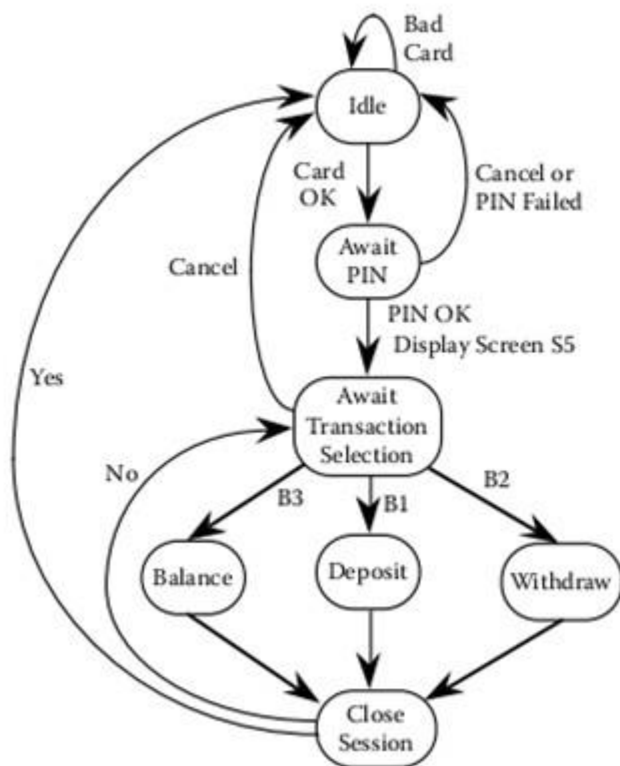
the world. It enjoys extensive CASE tool support as well as commercial training, and is described in numerous texts. The technique is based on three complementary models: function, data, and control. Here we use data flow diagrams for the functional models, entity/relationship models for data, and finite state machine models for the control aspect of the SATM system

That tool identifies external devices (such as the terminal doors) with lower case letters, and elements of the functional decomposition with numbers . The open and filled arrowheads on flow arrows signify whether the flow item is simple or compound. The portions of the SATM system shown here pertain generally to the personal identification number (PIN) verification portion of the system.

The Deft CASE tool distinguishes between simple and compound flows, where compound flows may be decomposed into other flows, which may themselves be compound



Upper level finite state machine. (5 M)



The upper level finite state machine in Figure divides the system into states that correspond to stages of customer usage.

5. Using Mc Cabe's strongly connected write the path/edge traversal. And explain cyclomatic complexity (8 M)

The cyclomatic complexity of a strongly connected graph is provided by the formula V(G) = e – n + p. The number of edges is represented by e, the number of nodes by n and the number of connected areas by p. If we apply this formula to the graph given in Figure 1.7, the number of linearly independent circuits is:

V(G) = e – n + p

= 11 – 7 + 1 = 5

If we now delete the edge from G to A, we can see that we have to identify 5 different independent paths to form our basis. An independent path is any path through the software that introduces at least one new set of processing statements or a new condition. To find these paths, McCabe developed a procedure known as the baseline method. The procedure works by starting at the source node. From here, the leftmost path is followed until the sink node is reached. If we take the example in Figure, this provides us with the path A, B, C, G. We then repeatedly retrace this path from the source node, but change our decisions at every node with out-degree ³ 2, starting with the decision node lowest in the path. For example, the next path would be A, B, C, B, C, G, as the decision at node C would be 'flipped'. The third path would then be A, B, E, F, G, as the next lowest decision node is B. Two important points should be made here. Firstly, if there is a loop, it only has to be traversed once, or else the basis will contain redundant paths. Secondly, it is possible for there to be more than one basis; the property of uniqueness is one not required.

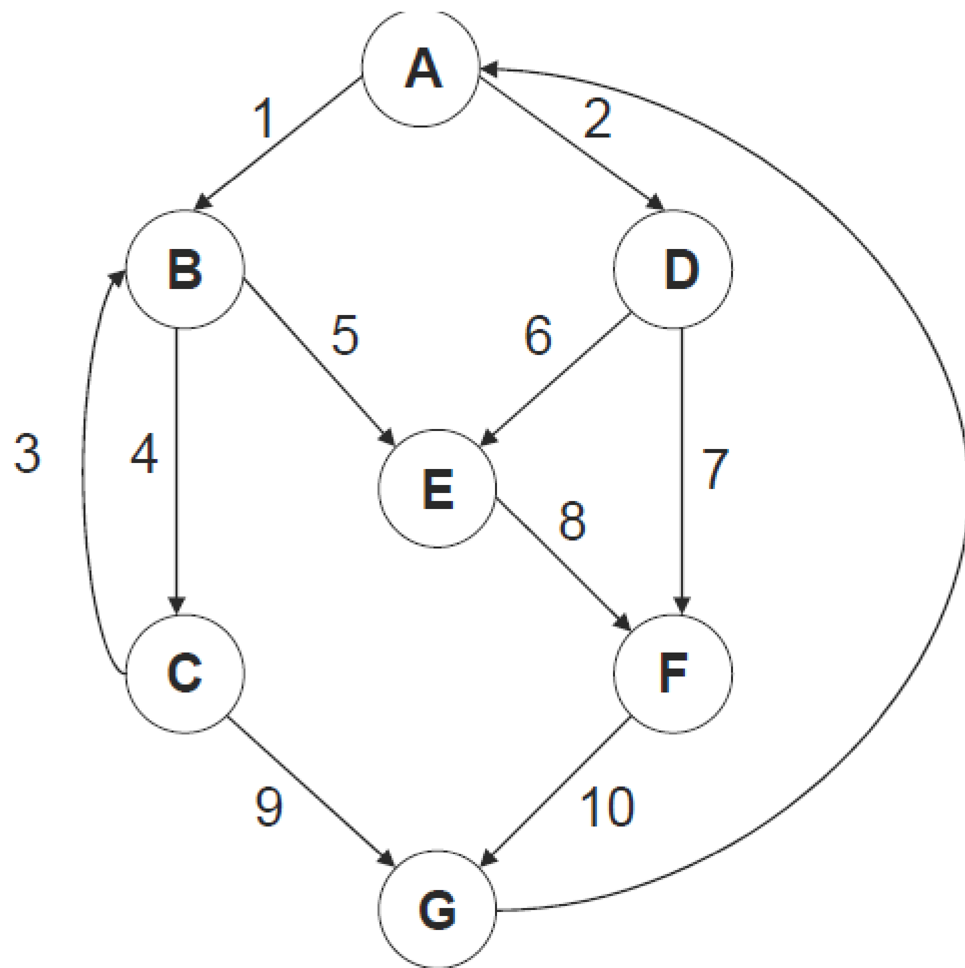
The five linearly independent paths of our graph are as follows:

Path 1: A, B, C, G.

Path 2: A, B, C, B, C, G.

Path 3: A, B, E, F, G.

Path 4: A, D, E, F, G.

	Path 5: A, D, F, G. 
6.	Define Predicate node, du paths, dc paths. Give du paths for stocks , locks, total locks and sales (10M)

A usage node USE(v, n) is a ***predicate use*** (denoted as P-use) iff the statement n is a predicate statement

A *definition-use* ***path*** with respect to a variable v (denoted du-path) is a (sub)path in PATHS(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 of the (sub)path

A *definition-clear* ***path*** with respect to a variable v (denoted dc-path) is a *definition-use* ***path*** (sub)path in PATHS(P) with initial and final nodes DEF (v, m) and USE (v, n) such that no other node in the (sub)path is a defining node of v. Testers should notice how these definitions capture the essence of computing with stored data values. Du-paths and dc-paths describe the flow of data across source statements from points at which the values are defined to points at which the values are used. Du-paths that are not definition-clear are potential trouble spots.

```
1  program lock-stock_and_barrel
2  const
3      lock_price = 45.0;
4      stock_price = 30.0;
5      barrel_price 25.0;
6  type
7       STRING_30 = string[30]; {Salesman's Name}
```

```
8    var
9         locks, stocks, barrels, num_locks, num_stocks,
10        num_barrels, salesman_index, order_index : INTEGER;
11        sales, commission : REAL;
12   salesman : STRING_30;
14     BEGIN {program lock_stock_and_barrel}
15    FOR salesman_index := 1 TO 4 DO
16     BEGIN
17        READLN(salesman);
18        WRITELN ('Salesman is ', salesman);
19       num_locks := 0;
20       num_stocks := 0;
21       num_barrels := 0;
22       READ(locks);
23       WHILE locks <> -1 DO
24        BEGIN
25           READLN (stocks, barrels);
26           num_locks := num_locks + locks;
27           num_stocks := num_stocks + stocks;
28           num_barrels := num_barrels + barrels;
29          READ(locks);
30        END; (WHILE locks)
31     READLN;
32      WRITELN('Sales for ',salesman);
33      WRITELN('Locks sold: ', num_locks);
34      WRITELN('Stocks sold: ', num_stocks);
35      WRITELN('Barrels sold: ', num_barrels);
36      sales := lock_price*num_locks + stock_price*num_stocks
           + barrel_price*num_barrels;
37     WRITELN('Total sales: ', sales:8:2);
38     WRITELN;
39     IF (sales > 1800.0) THEN
40      BEGIN
commission := 0.10 * 1000.0;
42        commission := commission + 0.15 * 800.0;
43        commission := commission + 0.20 * (sales-1800.0);
44      END;
45     ELSE IF (sales > 1000.0) THEN
46      BEGIN
commission := 0.10 * 1000.0;
48        commission := commission + 0.15*(sales - 1000.0);
49       END
50      ELSE commission := 0.10 * sales;
51       WRITELN('Commission is $',commission:6:2);
52    END; (FOR salesman)
53   END. {program lock_stock_and-barrel}
```

We have DEF(stocks, 25) and USE(stocks, 27), so the path <25, 27> is a du- path wrt (with respect to) stocks. Since there are no other defining nodes for stocks, this path is also definition-clear.

Two defining and two usage nodes make the locks variable more interesting: we have DEF(locks, 22), DEF(locks, 29), USE(locks, 23), and USE(locks, 26). These yield four du-paths:

p1 = <22, 23>

p2 = <22, 23, 24, 25, 26>

p3 = <29, 30, 23>

p4 = <29, 30, 23, 24, 25, 26>

Since there is only one defining node for sales, all the du-paths wrt sales must be definition-clear. They are interesting because they illustrate predicate and computation uses. The first three du-paths are easy:

p10 = <36, 37>

p11 = <36, 37, 38, 39>

p12 = <36, 37, 38, 39, 40, 41, 42, 43>

The du-paths for num_locks will lead us to typical test cases for computations. With two defining nodes (DEF(num_locks, 19) and DEF(num_locks, 26)) and three usage nodes (USE(num_locks, 26), USE(num_locks, 33), USE(num_locks, 36)), we might expect six du-paths. Let's take a closer look

p5= <19,20,21,22,23,24,25,26>

p6 = <19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33>

p7 = <19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36>

p8 = <26, 27, 28, 29, 30, 31, 32, 33>

p9 = <26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36>

p10 = <36, 37>

p11 = <36, 37, 38, 39>

p12 = <36, 37, 38, 39, 40, 41, 42, 43>

7. Discuss the usage of decision table method to device test cases with example of commission problem and triangle problem

Triangle Problem (5 M)

Conditions											

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	-	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	-	-	F	T	T	T	T	T	T	T	T
c4: a = b?	-	-	-	T	T	T	T	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b = c?	-	-	-	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1
a1: not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Case ID	**a**	**b**	**C**	**Expected Output**
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

Test Cases for the Commission Problem:(5 M)

Test Case	locks	Stocks	barrels	sales	commission
DT1	5	5	5	500	50
DT2	15	15	15	1500	175
DT3	25	25	25	2500	360

8. Explain i) Statement coverage (5 M) ii) Block coverage (5 M)

The statement coverage of T with respect to (P, R) is computed as Sc/(Se-Si) , where Sc is the number of statements covered, Si is the number of unreachable statements, and Se is the total number of statements in the program, i.e. the size of the coverage domain.
T is considered adequate with respect to the statement coverage criterion if the statement coverage of T with respect to (P, R) is 1.

The block coverage of T with respect to (P, R) is computed as Bc/(Be -Bi) , where Bc is the number of blocks covered, Bi is the number of unreachable blocks, and Be is the total number of blocks in the program, i.e. the size of the block coverage domain. T is considered adequate with respect to the block coverage criterion if the statement coverage of T with respect to (P, R) is 1.