USN

Internal Assessment Test 2 – June 2022

Sub:	**Software Testing**				Sub Code:	18IS62	Branch:	ISE
Date:	**08/06/2022**	Duration:	90 min’s	Max Marks: 50	Sem/Sec:	VI A, B, C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Describe weak normal, strong normal and weak robust, strong robust equivalence class testing, with a neat diagram.	10	CO1, CO2	L2
2a)	Explain decision table testing and generate test cases for commission problem using decision table	6	CO1, CO2	L2
2b)	Explain fault based testing, and mutation analysis.	4	CO2, CO3	L2

3	Consider the following program. Find the DU paths for the variables staff Discount, total Price, final Price, discount and price. Verify whether these DU paths are definition clear.
	`program Example()`
	`var staffDiscount, totalPrice, finalPrice, discount, price`
	`staffDiscount = 0.1`
	`totalPrice = 0`
	`input(price)`
	`while(price != -1) do`
	`    totalPrice = totalPrice + price`
	`    input(price)`
	`od`
	`print("Total price: " + totalPrice)`
	`if(totalPrice > 15.00) then`
	`    discount = (staffDiscount * totalPrice) + 0.50`
	`else`
	`    discount = staffDiscount * totalPrice`
	`fi`
	`print("Discount: " + discount)`
	`finalPrice = totalPrice - discount`
		10	CO3, CO4	L3

4	Explain McCabe's basis path testing with Triangle problem.	10	CO1	L2
5	Define DD path graph. Draw DD path graph for triangle program problem.	10	CO1	L2

6	Consider the following C function which encodes the string in the following manner. If the string character is + or - or *, it is replaced with space ' ‘, if it is uppercase character, it is replaced with lowercase. Other alphanumeric characters are simply copied in destination string. Draw the control flow graph for the program. Find out the statement coverage and node coverage % from control flow graph for the following test suite T0= {"test”, “test**ing", "test+-"}
	
	1. const char* encode (char *str) {
	2. int i = 0;
	3. char *str1=str;
	4. char en_str[25];
	5. while (str1[i] != '\0') {
		10	CO3, CO4	L3

	6. if(str1[i]=='*'|| str1[i]=='+'||str1[i] =='-')			

	7. en_str[i] =' '; 8. else if(str1[i]>=65 && str1[i]<=90) 9. en_str[i]=str1[i]+32; 10. else 11. en_str[i]=str1[i]; 12. i++; 13.} 14. en_str[i]='\0'; 15. return (en_str); }			

Faculty Signature **CCI Signature** **HOD Signature**

USN

Internal Assessment Test 2 – June 2022

Sub:	Software Testing-Scheme and Solutions					Sub Code:	18IS62	Branch:	ISE	
Date:	8/06/2022	Duration:	90 min's	Max Marks:	50	Sem/Sec:	VI C		OBE	
	Answer any FIVE FULL Questions							MARKS	CO	RBT
1	Describe weak normal, strong normal and weak robust, strong robust equivalence class testing, with a neat diagram.							[10]	CO1, CO2	L2

Weak normal [2.5 marks]

Explanation:1.5 marks

Diagram: 1 mark

Strong normal [2.5 marks]

Explanation:1.5 marks

Diagram: 1 mark

Weak robust [2.5 marks]

Explanation:1.5 marks

Diagram: 1 mark

Weak robust [2.5 marks]

Explanation:1.5 marks

Diagram: 1 mark

Weak Normal Equivalence Class Testing

• For the running example, we would end up with the three weak equivalence class test cases.

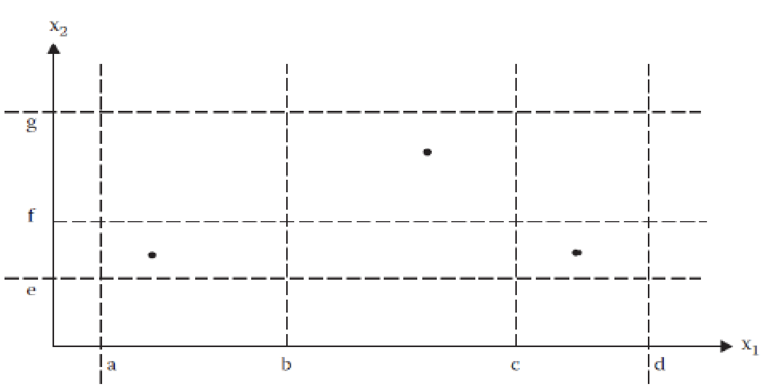
• These three test cases use one value from each equivalence class. The test case in the lower left rectangle corresponds to a value of x1 in the class [a, b), and to a value of x2 in the class [e, f).

• The test case in the upper center rectangle corresponds to a value of x1 in the class [b, c) and to a value of x2 in the class [f, g].

• The third test case could be in either rectangle on the right side of the valid values.

• There could be a problem with x1, or a problem with x2, or maybe an interaction between the two. This ambiguity is the reason for the "weak" designation.

• If the expectation of failure is low, as it is for regression testing, this can be an acceptable choice. When more fault isolation is required, the stronger forms,discussed next, are indicated.

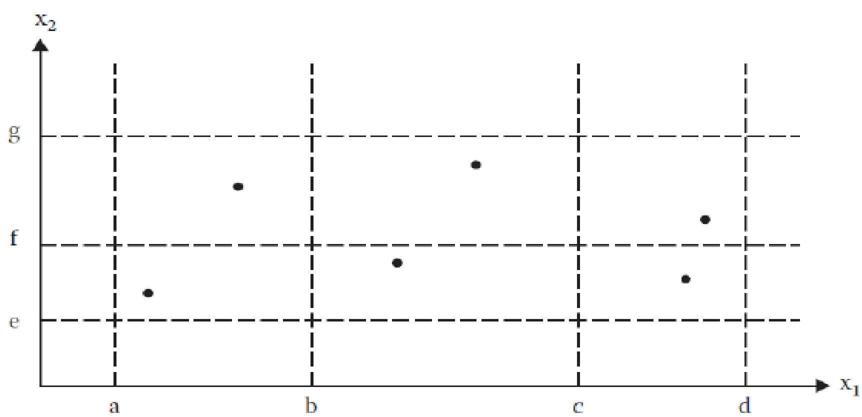


Strong Normal Equivalence Class Testing

•Strong equivalence class testing is based on the multiple fault assumption, so we need **test cases from each element of the Cartesian product of the equivalence classes.**

• The Cartesian product guarantees that we have a notion of "completeness" in two senses: we cover all the equivalence classes, and we have one of each possible combination of inputs. As we shall see from our continuing examples**, the key to "good" equivalence class testing is the selection of the equivalence relation.**

• Watch for the notion of inputs being "treated the same." Most of the time, equivalence class testing defines classes of the input domain. There is no reason why we could not define equivalence relations on the output range of the program function being tested; in fact, this is the simplest approach for the triangle problem.

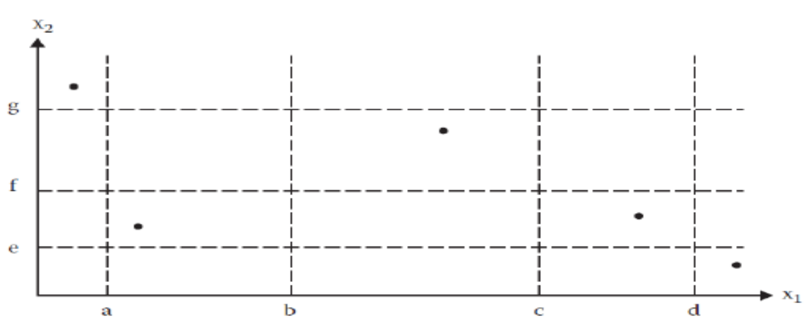


Weak Robust Equivalence Class Testing

• The robust part comes from consideration of invalid values, and the weak part refers to the single fault assumption. The process of weak robust equivalence class testing is a simple extension of that for weak normal equivalence class testing—pick test cases such that each equivalence class is represented.

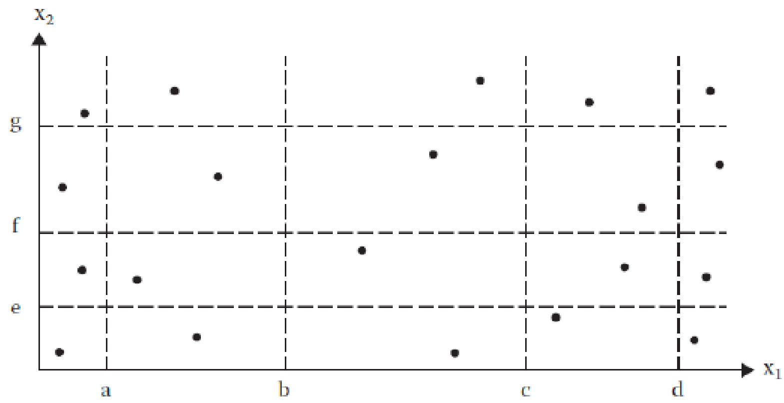
• **The two additional test cases cover all four cla**sses of invalid values. The process is similar to that for boundary value testing:

1. For valid inputs, use one value from each valid class (as in what we have called weak normal
equivalence class testing). (Note that each input in these test cases will be valid.)
2. For invalid inputs, a test case will have one invalid value and the remaining values will all be
valid. (Thus, a "single failure" should cause the test case to fail.)



Strong Robust Equivalence Class Testing

• The robust part comes from consideration of invalid values, and the strong part refers to the multiple fault assumption. We obtain test cases from each element of the Cartesian product of all the equivalence classes, both valid and invalid.



2a) Explain decision table testing and generate test cases for commission problem using decision table. [6] CO1, CO2 L2

Explanation: 1Mark
Test case: 5Marks

Decision Table Testing

- To identify test cases with decision tables, we interpret conditions as inputs and actions as outputs. Sometimes conditions end up referring to equivalence classes of inputs, and actions refer to major functional processing portions of the item tested.
- The rules are then interpreted as test cases.
- Decision table have some assurance that we will have a comprehensive set of test cases. Several techniques that produce decision tables are more useful to testers.

Test Cases for Commission Problem

RULES		R1	R2	R3	R4	R5	R6	R7	R8	R9
Conditions	C1: Locks = -1	T	F	F	F	F	F	F	F	F
	C2: 1 ≤ Locks ≤ 70	-	T	T	T	T	F	F	F	F
	C3: 1 ≤ Stocks ≤ 80	-	T	T	F	F	T	T	F	F
	C4: 1 ≤ Barrels ≤ 90	-	T	F	T	F	T	F	T	F
Actions	a1: Terminate the input loop	X								
	a2: Invalid locks input						X	X	X	X
	a3: Invalid stocks input				X	X			X	X
	a4: Invalid barrels input			X		X		X		X
	a5: Calculate total locks, stocks and barrels		X	X	X	X	X	X	X	
	a6: Calculate Sales	X								
	a7: proceed	X								

RULES	R1	R2	R3	R4
C1 : Sales = 0	T	F	F	F
C1 : Sales > 0 AND Sales ≤ 1000		T	F	F
C2 : Sales > 1001 AND sales ≤ 1800			T	F
C3 : sales ≥1801				T
A1 : Terminate the program	X			
A2 : comm= 10%*sales		X		
A3 : comm = 10%*1000 + (sales-1000)*15%			X	
A4 : comm = 10%*1000 + 15% * 800 + (sales-1800)*20%				X

2b) Explain fault based testing, and mutation analysis. [4] CO2, CO3 L2

<u>Fault Based Testing [2 marks]</u>

Explanation:2 marks

<u>Mutation Analysis [2 marks]</u>

Explanation:2 marks

<u>Fault Based Testing</u>

• A model of potential program faults is a valuable source of information for evaluating and designing test suites.

• Some fault knowledge is commonly used in functional and structural testing, for example when identifying singleton and error values for parameter characteristics in category-partition testing or when populating catalogs with erroneous values, but a fault model can also be used more directly.

• Fault-based testing uses a fault model directly to hypothesize potential faults in a program under test, as well as to create or evaluate test suites based on its efficacy in detecting those hypothetical faults.

• **The basic concept of fault-based testing is to select test cases that would distinguish the program under test from alternative programs that contain hypothetical faults.**

• This is usually approached **by modifying the program under test to actually produce the hypothetical faulty programs.**

<u>Mutation analysis</u>

• Mutation analysis is the most common form of software fault-based testing. A fault model is used to produce hypothetical faulty programs by creating variants of the program under test. Variants are created by "seeding" faults, that is, by making a small change to the program under test following a pattern in the fault model.

•The patterns for changing program text are called mutation operators, and each variant program is called a mutant.

• We say a mutant is valid, if it is syntactically correct. A mutant obtained from the program by substituting while for switch in the statement at line 13 would not be valid, since it would result in a compile-time error.

3 Consider the following program. Find the DU paths for the variables staff Discount, total Price, final Price, discountand price. Verify whether these DU paths are definition clear. 10 CO3, CO4 L3

```
1  program Example()
2  var staffDiscount, totalPrice, finalPrice, discount, price
3  staffDiscount = 0.1
4  totalPrice = 0
5  input(price)
6  while(price != -1) do
7      totalPrice = totalPrice + price
8      input(price)
9  od
10 print("Total price: " + totalPrice)
11 if(totalPrice > 15.00) then
12     discount = (staffDiscount * totalPrice) + 0.50
13 else
14     discount = staffDiscount * totalPrice
15 fi
16 print("Discount: " + discount)
17 finalPrice = totalPrice - discount
```

Staff Discount: 2Marks
Total Price: 2Marks
Final Price: 2Marks
Discount: 2Marks
Price: 2Marks

DU path for staff discount

P1 (3, 12) = <3, 4, 5, 6, 7, 8, 9, 10, 11, 12> is definition clear
P2 (3, 14) = <3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14> is not definition clear

DU path for total price

P3 (4, 7) = <4, 5, 6, 7> is definition clear
P4 (4, 10) = <4, 5, 6, 7, 8, 9, 10> is not definition clear
P5 (7, 6) = <7, 8, 9, 6> is definition clear
P6 (7, 7) = <7, 8, 9, 6, 7> is not definition clear
P7 (7, 10) = <7, 8, 9, 6, 10> is definition clear
P8 (7, 11) = <7, 8, 9, 6, 10, 11> is definition clear
P9 (7, 12) = <7, 8, 9, 6, 10, 11, 12> is definition clear
P10 (7, 14) = <7, 8, 9, 6, 10, 11, 12, 13, 14> is definition clear

DU path for final price

P11 (17, 17) = <17, 17> is definition clear

DU path for discount

P12 (12, 16) = <12, 13, 14, 15, 16> is not definition clear
P13 (12, 17) = <12, 13, 14, 15, 16, 17> is not definition clear
P14 (12, 16) = <12, 13, 14, 15, 16> is not definition clear
P15 (14, 16) = <14, 15, 16> is definition clear
P16 (14, 17) = <14, 15, 16, 17> is definition clear

DU path for price

P17 (5, 6) = <5, 6> is definition clear
P18 (5, 7) = <5, 6, 7> is definition clear
P19 (8,6) = <8, 9, 6> is definition clear
P20 (8, 7) = <8, 9, 6, 7> is definition clear

4	Explain McCabe's basis path testing with Triangle problem. **Explanation: 2Marks** **Diagram: 5Marks** **Basis Paths Tables: 2Marks**	[10]	CO1	L2

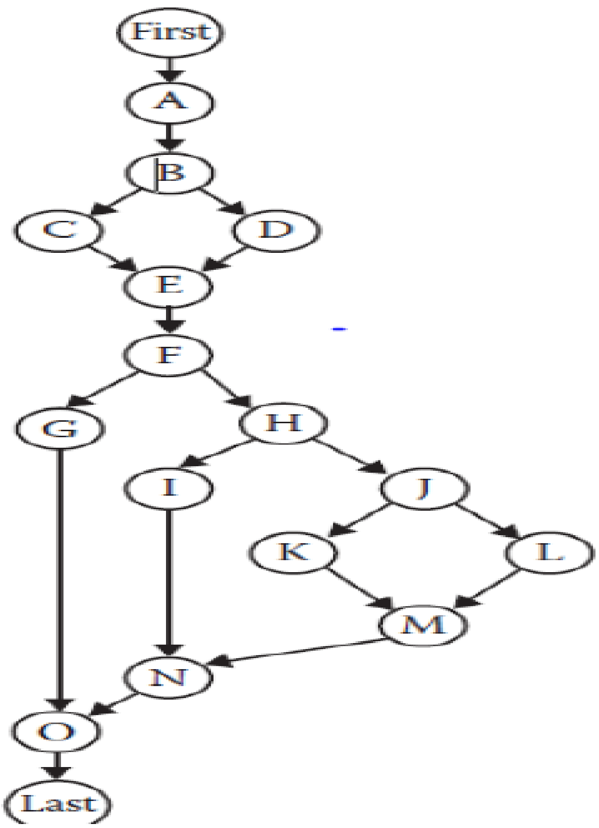
McCabe's Basis Path Method:

• **McCabe's** view is: There are two soft spots

1. Testing only the set of basis paths is sufficient.
2. Program paths look like a vector space.

• McCabe's example that the path A, B, C, B, C, B, C, G is the linear combination 2p2 – p1 is very unsatisfactory.

• To get a better understanding of these problems, we will go back **to the triangle program example. DD-Path of triangle given below.**



- We begin a baseline path corresponding path with scalene Triangle.
- Basis Path: Path with highest Decision tables
- Flip at node with outdegree=2
- flip at node B
- flip at node F
- flip at node H
- flip at node J

Original	p1: A–B–C–E–F–H–J–K–M–N–O–Last	Scalene
Flip p1 at B	p2: A–B–D–E–F–H–J–K–M–N–O–Last	Infeasible
Flip p1 at F	p3: A–B–C–E–F–G–O–Last	Infeasible
Flip p1 at H	p4: A–B–C–E–F–H–I–N–O–Last	Equilateral
Flip p1 at J	p5: A–B–C–E–F–H–J–L–M–N–O–Last	Isosceles

• We are dealing with **code-level dependencies, which are incompatible with the latent assumption that basis paths are independent.**
• McCabe's procedure successfully identifies basis paths that are topologically independent.

p1: A–B–C–E–F–H–J–K–M–N–O–Last	Scalene
p6: A–B–D–E–F–G–O–Last	Not a triangle
p4: A–B–C–E–F–H–I–N–O–Last	Equilateral
p5: A–B–C–E–F–H–J–L–M–N–O–Last	Isosceles

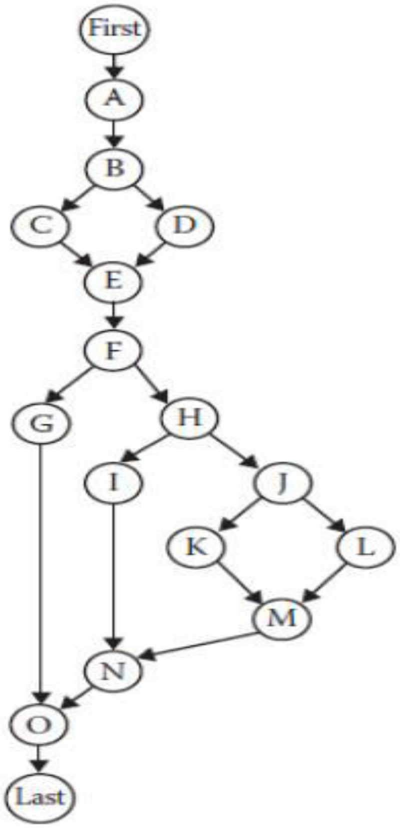
5 Define DD path graph. Draw DD path graph for triangle program problem. [10] CO1 L2

Explanation: 3Marks
Diagram: 5Marks
DD path Table: 2Marks

Definition:

Given a program written in an imperative language, its DD-path graph is the directed graph in which nodes are DD-paths of its program graph, and edges represent control flow between successor DD-paths.

DD Path Graph for Triangle Problem



DD path Table

Nodes	DD path	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	G	3
15	H	4
16	I	3
17	J	4
18	K	4
19	L	3
20	M	3
21	N	4
22	O	3
23	Last	2

6	Consider the following C function which encodes the string in the following manner. If the string character is + or - or *, it is replaced with space ' ‘, if it is uppercase character, it is replaced with lowercase. Other alphanumeric characters are simply copied in destination string. Draw the control flow graph for the program. Find out the statement coverage and node coverage % from control flow graph for the following test suite T0= {"test”, “test**ing", "test+-"}	10	CO3, CO4	L3

```
1. const char* encode (char *str) {
2. int i = 0;
3. char *str1=str;
4. char en_str[25];
5. while (str1[i] != '\0') {

  6. if(str1[i]=='*'|| str1[i]=='+'||str1[i] =='-')
  7. en_str[i] =' ';
  8. else if(str1[i]>=65 && str1[i]<=90)
  9. en_str[i]=str1[i]+32;
 10. else
  11. en_str[i]=str1[i];
  12. i++;
  13.}
  14. en_str[i]='\0';
  15. return (en_str);
     }
```

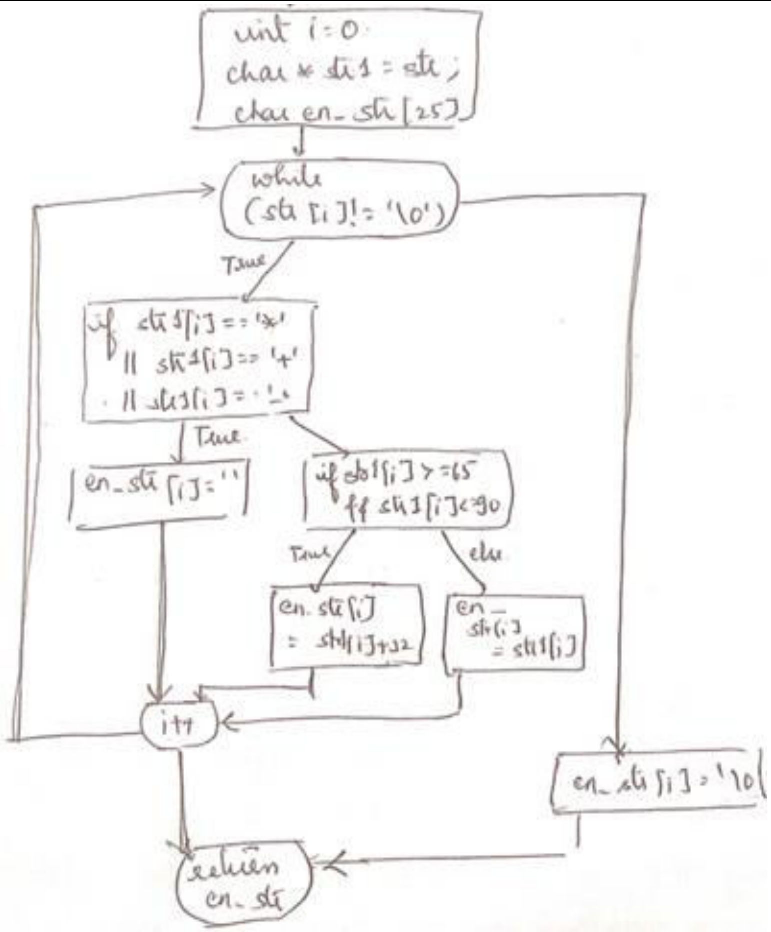
Statement Coverage and Node Coverage %: 3Marks
Control Flow Graph: 7Marks

<u>Statement Coverage and Node Coverage %</u>

Number of nodes = 2
Number of statements = 12

Given test suite = <’’test’’, test ** ing”, “test+-“}
Test suite does not contain special symbols, upper case letters
It will not visit F node
Statement coverage = 11/12 = 91.6%
Node coverage = 8/9 = 88.8%

<u>Control flow graph</u>

	int i=0
	char * str1 = str;
	char en_str[25]
	while (str[i]!='\0')
	True
	if str1[i]=='x' || str1[i]=='+' || str1[i]=='_'
	True
	en_str[i]=' '
	if str1[i]>=65 && str1[i]<=90
	True
	else
	en_str[i] = str1[i]+32
	en_str[i] = str1[i]
	i++
	en_str[i]='\0'
	return en_str			