USN

Internal Assessment Test 1 – May 2021

Sub:	Software Testing-Scheme and Solutions			Sub Code:	18CS62/17 CS62	Branch:	ISE
Date:	21/05/2021	Duration:	90 min's	Max Marks: 50	Sem/Sec:	VI A,B&C	OBE

Answer any FIVE FULL Questions — MARKS | CO | RBT

1a) Differentiate Error, Fault, and Failure with example — [4] CO1 L2

Definition:2 marks
Example: 2 marks

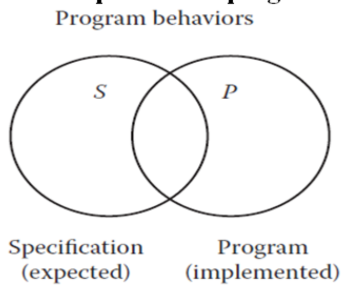
Error	Fault	Failure
Human Mistake, or bugs	A fault is the result of an error. It is more precise to say that a fault is the representation of an error, where representation is the mode of expression, such as narrative text, dataflow diagrams, hierarchy charts, source code, and so on.	When fault code is executed failure will occur.
Error in business logic in Requirements. discount 15% instead of 10% for purchase Rs. 10000	Fault in SRS and in discount calculation	Wrong reduction value for customer

1b) What is the use of Venn diagram in testing? Explain with diagram. — [6] CO1 L2

Point 1: 1mark
Two diagramsand Explanation: 2.5+2.5 = 5 marks

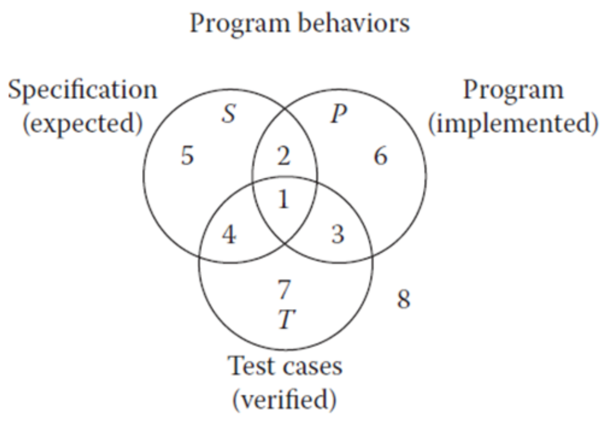
- Venn Diagrams are helpful in identifying the test cases. Venn Diagrams helps to find certain specified behaviors have not been programmed and certain programmed (implemented) behaviors have not been specified. These correspond to faults of commission and to errors that occurred after the specification was complete.
-

Specified and implemented program behaviors



Explanation:1.5 marks

Specified, implemented, and tested behaviors



Explanation: 1.5 marks

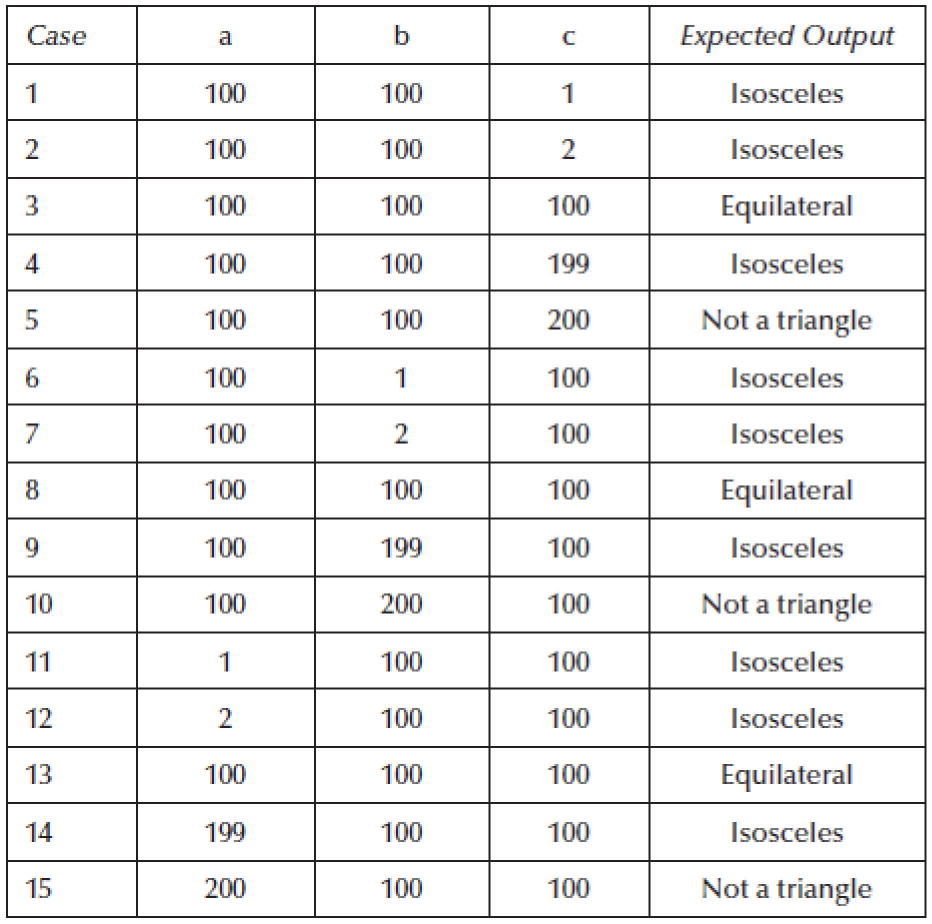
2a)	Compare specification testing with code based testing **4 points: 4 marks**	[4]	CO2	L2

Specification Testing	Code Based Testing
This is also called black box testing, in which the content (implementation) of the black box is not known, and the function of the black box is understood completely in terms of its inputs and outputs	it is sometimes called white box (or even clear box) testing. The essential difference is that the implementation(of the black box) is known and used to identify test cases.
the only information used is the specification of the software.	code-based testing uses the program source code (implementation) as the basis of test case identification.
Advantages (1) they are independent of how the software is implemented, so if the implementation changes, the test cases are still useful; and (2) test case development can occur in parallel with the implementation, thereby reducing the overall project development interval.	The ability to "see inside" the black box allows the tester to identify test cases on the basis of how the function is actually implemented.
Disadvantages: specification based test cases frequently suffer from two problems: <u>significant redundancies</u> may exist among test cases, compounded by the <u>possibility of gaps of untested software</u>	High Test case coverage.ess Redundancy. Gaps are covered

2b)	Write and explain the improved version of Triangle problem with generated test cases using Normal Boundary value analysis **Program: 3Marks** **Test cases: 2 marks** **Explanation: 1 Mark**	[6]	CO1	L3

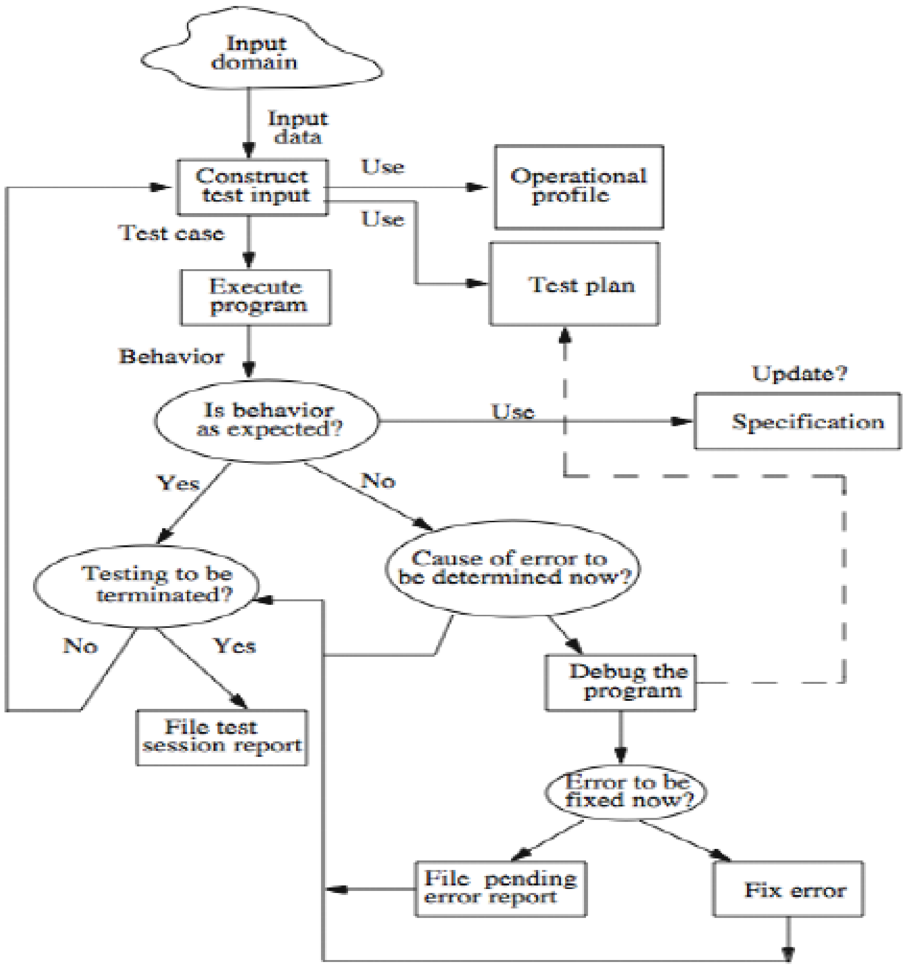
```
Program triangle3'
Dim a, b, c As Integer
Dim c1, c2, c3, IsATriangle As Boolean
'Step 1: Get Input
Do
   Output("Enter 3 integers which are sides of a triangle")
   Input(a, b, c)
   c1 = (1 ≤ a) AND (a ≤ 300)
   c2 = (1 ≤ b) AND (b ≤ 300)
   c3 = (1 ≤ c) AND (c ≤ 300)
   If NOT(c1)
        Then Output("Value of a is not in the range of permitted values")
   EndIf
   If NOT(c2)
        Then Output("Value of b is not in the range of permitted values")
   EndIf
   If NOT(c3)
        ThenOutput("Value of c is not in the range of permitted values")
   EndIf
Until c1 AND c2 AND c3
Output("Side A is",a)
Output("Side B is",b)
Output("Side C is",c)
'Step 2: Is A Triangle?
If (a < b + c) AND (b < a + c) AND (c < a + b)
   Then IsATriangle = True
   Else IsATriangle = False

EndIf
'Step 3: Determine Triangle Type
If IsATriangle
    Then If (a = b) AND (b = c)
             Then Output ("Equilateral")
             Else If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
                      Then Output ("Scalene")
                      Else Output ("Isosceles")
                  EndIf
         EndIf
    Else Output("Not a Triangle")
EndIf
End triangle3
```

Table 5.1 Normal Boundary Value Test Cases



Case	a	b	c	*Expected Output*
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	100	100	Equilateral
9	100	199	100	Isosceles
10	100	200	100	Not a triangle
11	1	100	100	Isosceles
12	2	100	100	Isosceles
13	100	100	100	Equilateral
14	199	100	100	Isosceles
15	200	100	100	Not a triangle

3 (a)	Explain Test and Debug Cycle with a neat diagram. **Diagram: 3marks** **Explanation: 3 marks**	[6]	CO1	L2



(b) Compute the Cyclomatic complexity of the following code and explain how to compute this. [4] CO1 L3

```
     void fun1(int n)
        {
         int x=5;
         if( (x<= n)
          x++;
          }
```

Answer: 2 marks
Explanation: 2 marks

Answer: 2
Form Main function is called. + 1 added to complexity and Function contains one conditional statement. +1 added to complexity. So total- 2.

4(a) Write the test cases for the C function which takes two integers as input and finds the maximum of the two integers using Robust Boundary Value analysis, and Worst case Boundary Value analysis. Assume the inputs are in the range of 1 to 35000. [3+3] CO1 L3

No of Inputs: 2 integers in the range 1 to 35000**[0.5 marks]**
{min, min+, nom, max-, max} ={1,2, 17500, 34999,35000}
Robust Boundary Value: {0,1,2,15000,34999,35000,35001}
Test Cases: 6n+1=6*2+1=13**[2.5 marks]**

S.No	**a**	**b**	**Output**
1	17500	0	Invalid Input
2	17500	1	17500
3	17500	2	17500
4	0	17500	Invalid Input

5	1	17500	Invalid Input
6	2	17500	17500
7	17500	34999	34999
8	17500	35000	35000
9	17500	17500	17500
10	34999	17500	34999
11	35000	17500	35000
12	35001	17500	35001
13	17500	17500	17500

Worst case Boundary Value analysis [0.5 marks]

{min, min+, nom, max-, max}={1,2, 17500, 34999,35000}

Number of Test cases:= 5^n =5*5=25

[2.5 marks]If minimum 10 test cases if they write also give 2.5 marks

S.No	a	b	Output
1	1	1	Invalid Input
2	1	2	17500
3	1	17500	17500
4	1	34999	Invalid Input
5	1	35000	Invalid Input
6	2	1	17500
7	2	2	34999
8	2	17500	35000
9	2	34999	17500
10	2	35000	34999
11	17500	1	35000
12	17500	2	35001
13	17500	17500	17500
14	17500	34999	Invalid Input
15	17500	35000	17500
16	34999	1	17500
17	34999	2	Invalid Input
18	34999	17500	Invalid Input
19	34999	34999	17500
20	34999	35000	34999
21	35000	1	35000
22	35000	2	17500
23	35000	17500	34999
24	35000	34999	35000
25	35000	35000	35001

(b) Define the following software quality attributes [04] CO1 L2

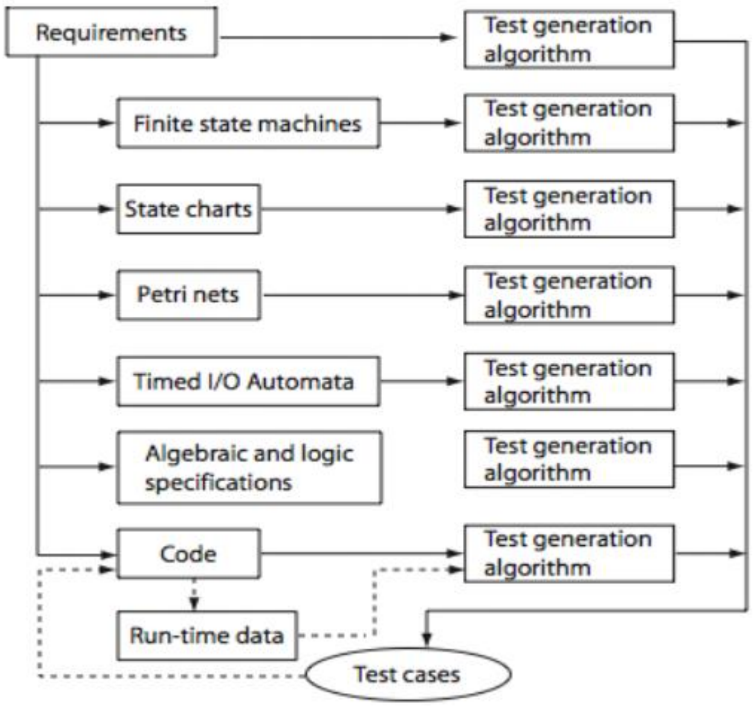
a) Reliability b) Consistency

2*2=4 marks

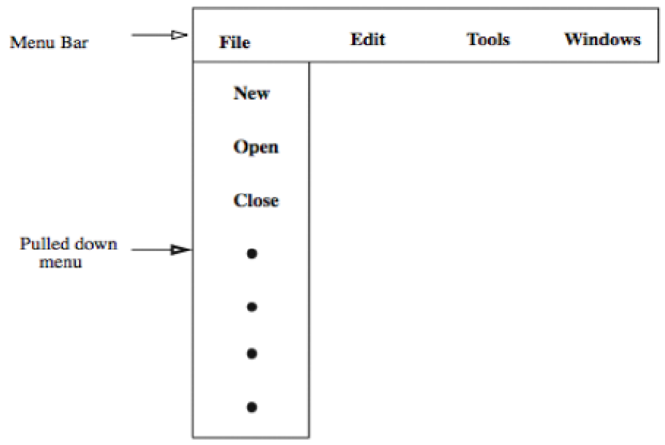
Reliability: Probability of failure of a software product with respect to a given operational profile in a given environment. is the probability of failure free operation of software in its intended environment.

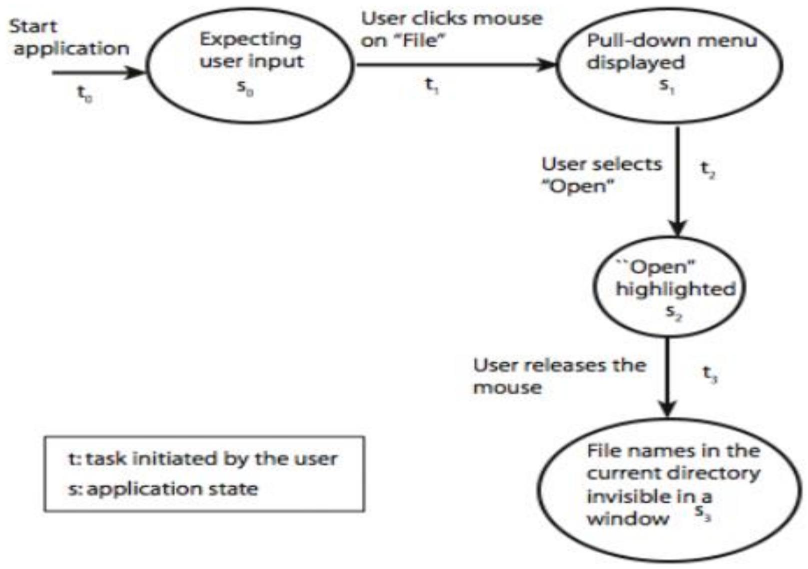
Consistency:

refers to adherence to a common set of conventions and assumptions. For example, all buttons in the user interface might follow a common color coding convention.

	An example of inconsistency would be when a database application displays the date of birth of a person in the database without regard for the user's preferences.			
5 (a)	Explain Test generation strategies with diagram • The tests are generated using a mix of formal and informal methods either directly from the requirements document serving as the source. • In more advanced test processes, requirements serve as a source for the development of formal models. • Several strategies are there for test case generation • These techniques identify input variables and use formal techniques for test generation and cause effect graphing. • Another way is use of model based testing • They need subset of requirements to be modeled using a formal notation which is called as specification. The tests are generated from specification using FSMs, Statecharts, Petri Nets and Timed I/O Automata notations for modeling. • Unified modeling language can also used for modeling the requirements into proper specification for test case generation. • Model can also be built using predicate Logic and algebraic languages. Each model has its own strengths and weaknesses  Requirements Test generation algorithm Finite state machines State charts Petri nets Timed I/O Automata Algebraic and logic specifications Code Run-time data Test cases • Code based techniques can be used to generate tests, or modify existing ones, to generate new tests that force a condition to evaluate to true or false. • Two techniques: Program mutation and control flow coverage techniques	[5]	CO1	L2
(b)	Explain how to write a Oracle program for GUI with example. Draw the state diagram for the same. **Example with diagram 2 marks** **State Diagram: 2 marks**	[5]	CO1	L3

Consider a menu driven application.




Start application
t_0
Expecting user input s_0
User clicks mouse on "File"
t_1
Pull-down menu displayed s_1
User selects "Open"
t_2
"Open" highlighted s_2
User releases the mouse
t_3
File names in the current directory invisible in a window s_3
t: task initiated by the user
s: application state

6	Explain in detail about Normal Boundary value analysis, and Robust Boundary value analysis with input domain diagrams.	[10]	CO2	L2

Normal Boundary value analysis[5 marks]

Explanation:2.5 marks

Diagram: 1.5 marks

Diagram explanation: 1 mark

Robust Boundary value analysis[5 marks]

Explanation:2.5 marks

Diagram: 1.5 marks

Diagram explanation: 1 mark

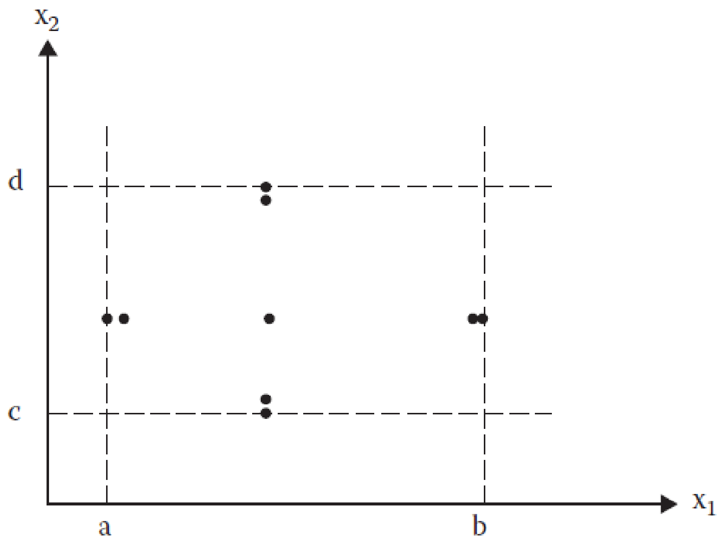
Normal Boundary value analysis

- The basic idea of boundary value analysis is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and at their maximum.
- **values are min, min+, nom, max- and max; The robust forms add two values, min– and max+.**
- The next part of boundary value analysis is based on a critical assumption; it's known as the "single fault" assumption in reliability theory. This says that failures are only rarely the result of the simultaneous occurrence of two (or more) faults.

- Thus the boundary **value analysis test cases are obtained by holding the values of all but one variable at their nominal values, and letting that variable assume its extreme values.**
- The boundary value analysis test cases for our function F of two variables are:

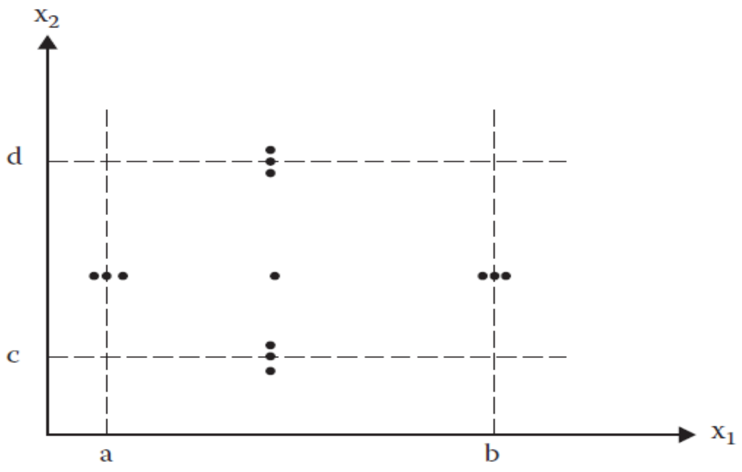
{<x1nom, x2min>, <x1nom, x2min+ >,<x1nom, x2nom>,<x1nom, x2max- >, <x1nom, x2max>, <x1min, x2nom >, <x1min+, x2nom >, <x1max-, x2nom >, <x1max, x2nom > }

- These are illustrated in the following Figure .



<u>Robust Boundary value analysis</u>

- Robust boundary value testing is a simple extension of normal boundary value testing: in addition to the five boundary value analysis values of a variable, we see what happens when the extrema are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–).
- Robustness test cases for our continuing example are shown in Figure.



- Most of the discussion of boundary value analysis applies directly to robustness testing, especially **the generalizations and limitations**. The most interesting part of robustness testing is not with the inputs but **with the expected outputs**.
- The main value of robustness testing is that it forces **attention on exception handling.** With <u>strongly typed languages, robustness testing may be very awkward</u>.
- Pascal, for example, if a variable is defined to be within a certain range, values

	outside that range result in run-time errors that abort normal execution. • This raises an interesting question of implementation philosophy: is it better to perform explicit range checking and use exception handling to deal with “robust values,” or is it better to stay with strong typing? The <u>exception handling choice mandates robustness testing.</u>			