Scheme of Evaluation
Internal Assessment Test 1 – March 2019

Sub:	Software Testing							**Code:**	15IS63
Date**:**	06/03/2019	Duration:	90mins	Max Marks:	50	**Sem:**	VI	**Branch:**	ISE

Note: Answer Any Five Questions

Question #		**Description**	**Marks Distribution**		**Max Marks**
1	a)	**Define the following**: i) Error ii) Fault iii) failure iv) incident v) Reliability vi) Operational profile	1 M 1M 1M 1M 1M 1M	6 M	10 M
	b)	Explain the Test generation strategies with a diagram • Model based testing • Specification based testing • Code based testing • Diagram	1 M 1 M 1 M 1M	4 M	
2		**With a neat diagram, explain the SATM system.** • SATM GUI • Diagram for 15 transaction screens • Explanation of working of ATM with 3 types of transactions (withdrawal, deposit, balance enquiry)	1 M 2 M 7 M	10 M	10 M
3	a)	**Briefly explain testing using Venn Diagram** • Venn Diagram • Identifying each region and explanation of all regions	1M 4M	5 M	10 M
	b)	**Differentiate between functional testing and structural testing?** • Functional testing • Structural Testing	2.5M 2.5M	5 M	

4		**Briefly explain weak normal and strong robust equivalence class testing and apply them to derive test cases for triangle problem** • Weak normal ECT with diagram • Strong Robust ECT with diagram • Test cases for triangle problem	3M 3M 4M	10 M	10 M
5	a)	**Write a short note on random testing** • Explanation of random testing • Formula for random testing • Example	3 M 1 M 2 M	6 M	10 M
	b)	**Discuss what typical test case information should include**. • Listing of all fields	4 M	4 M	
6	a)	**Justify the usage of boundary value analysis with function of two variables and highlight the limitations of BVA**. • General BVA • Worst case BVA • Robust BVA • Robust worst case BVA • Limitations	2M 2M 2M 2M 2M	10 M	10 M
7	a)	**Define oracle and explain its construction with example** • Oracle Definition • Hvideo application example • Diagram	1M 3M 1M	5 M	10 M
	b)	**Explain with diagram the currency converter.** • GUI • Explanation of application • Testing the GUI	1M 3M 1M	5 M	

IAT-1 Solution
Software Testing (15IS63)
March 2018-19

1 a) Define the following

i. **Error**: People make errors. A good synonym is "mistake". When people make mistakes while coding, we call these mistakes "bugs". Errors tend to propagate; a requirements error may be magnified during design, and amplified still more during coding

ii. **Fault**: 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. "Defect" is a good synonym for fault; so is "bug". Faults can be elusive.

iii. **Failure**: A failure occurs when a fault executes. Two subtleties arise here: one is that failures only occur in an executable representation, which is usually taken to be source code, or more precisely, loaded object code. The second subtlety is that this definition relates failures only to faults of commission

iv. **Incident**: When a failure occurs, it may or may not be readily apparent to the user (or customer or tester). An incident is the symptom(s) associated with a failure that alerts the user to the occurrence of f**a**ilure.

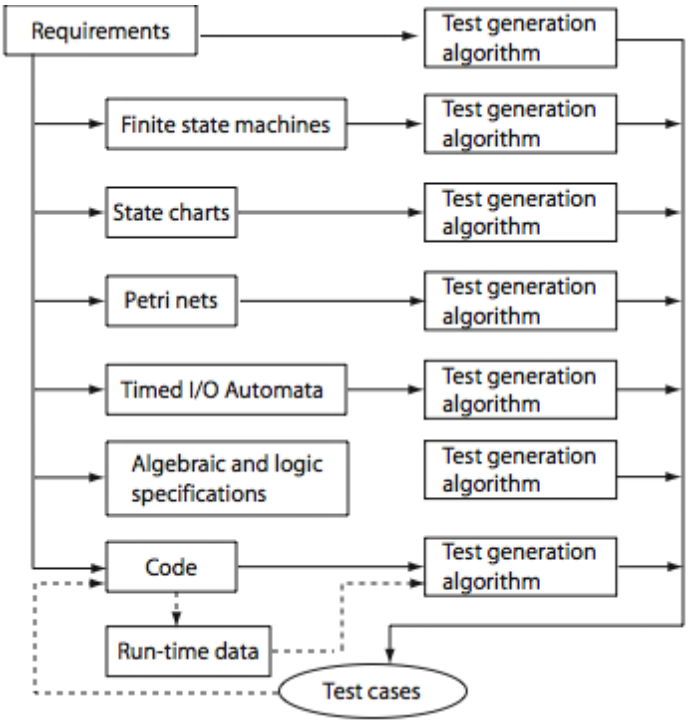
v. **Software reliability**: is the probability of failure free operation of software over a given time interval and under given conditions.

vi. An **operational profile** is a numerical description of how a program is used

b) Explain the Test generation strategies with a diagram

Any form of test generation uses a source document. In the most informal of test methods, the source document resides in the mind of the tester who generates tests based on a knowledge of the requirements.
In several commercial environments, the process is a bit more formal. 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.

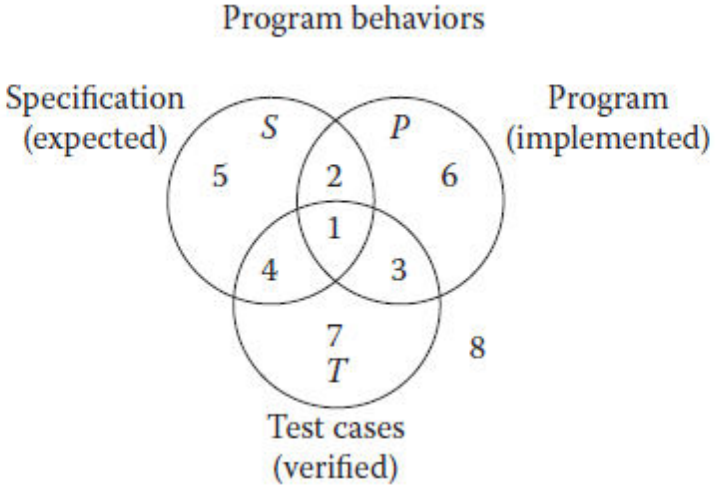
- **Model based**: require that a subset of the requirements be modeled using a formal notation (usually graphical). Models: Finite State Machines, Timed automata, Petri net, etc.
- **Specification based**: require that a subset of the requirements be modeled using a formal mathematical notation.
- **Code based**: generate tests directly from the code.



➢

3 a) Briefly explain testing using Venn diagram

Testing is fundamentally concerned with behavior; and behavior is orthogonal to the structural view common to software (and system) developers. A quick differentiation is that the structural view focuses on "what it is" and the behavioral view considers "what it does



- 2, 5: Specified behavior that are not tested
- 1, 4: Specified behavior that are tested
- 3, 7: Test cases corresponding to unspecified behavior
- 2, 6:Programmed behavior that are not tested

- 1, 3:Programmed behavior that are tested
- 4, 7:Test cases corresponding to un-programmed behaviors

If there are specified behaviors for which there are no test cases, the testing is incomplete. If there are test cases that correspond to unspecified behaviors Either such test cases are unwarranted, or
Specification is deficient: It also implies that testers should participate in specification and design reviews

b) Differentiate between functional testing and structural testing?

Functional testing is based on the view that any program can be considered to be a function that maps values from its input domain to values in its output range. This leads to the term black box testing in which the content (implementation) of a black box is not known, and the function of the black box is understood completely in terms of its inputs and outputs.

With the functional approach to test case identification, the only information that is used is the specification of the software. There are two distinct advantages to functional test cases: they are independent of how the software is implemented, so if the implementation changes, the test cases are still useful, and test case development can occur in parallel with the implementation, thereby reducing overall project development interval. On the negative side functional test cases frequently suffer from two problems:

1. Redundancies
2. Gaps in tested software.

Structural Testing

Structural testing is the other fundamental approach to test case identification. To contrast it with Functional Testing, it is sometimes called White Box (or even Clear Box) Testing. The clear box metaphor is probably more appropriate, because the essential difference is that the implementation (of the Black Box) is known and used to identify test cases.

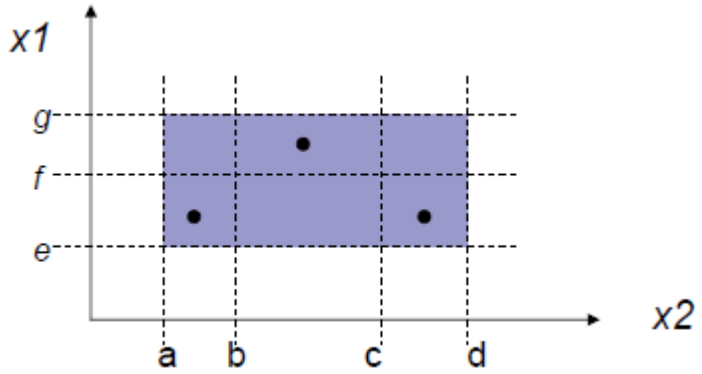
4) Briefly explain weak normal and strong robust equivalence class testing and apply them to derive test cases for triangle problem

Consider a function F, of two variables x1 and x2

- x1 and x2 have the following boundaries and intervals within boundaries:
  $a \leq x1 \leq d$, with intervals [a, b) [b, c), [c, d)
  $e \leq x2 \leq g$, with intervals [e, f) [f, g)
- [= closed interval, (= open interval

In **weak normal equivalence class**:

- One variable from each equivalence class
- Values identified in systematic way



Robust Equivalence Class Testing

- Robust - consideration of invalid values.

Two problems with robust ECT

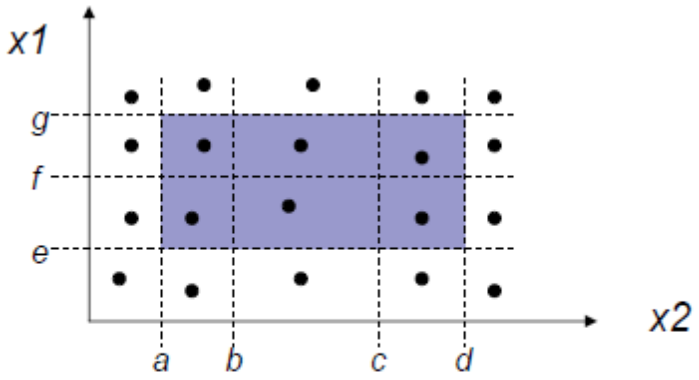
- Specification (expected output for invalid TC?)
- Strongly typed languages (eliminate need), Traditional equivalence class testing (FORTRAN, COBAL) – errors common

Weak Robust Equivalence Class Testing

- Valid inputs – weak normal ECT
- Invalid inputs – each TC has one invalid value, single fault should cause failure.

Strong Robust Equivalence Class Testing

- Combination of both robust and strong



Triangle Problem
Four possible outputs – NotA-Triangle, Scalene, Isosceles and Equilateral.
R1 = {<a,b,c> : the triangle with sides a.b and c is equilateral}
R2 = {<a,b,c> : the triangle with sides a,b and c is isosceles}
R3 = {<a,b,c> : the triangle with sides a,b and c is isosceles}
R4 = {<a,b,c> : sides a,b and c do not form a triangle}

Weak Normal Test cases

Test Case	a	b	c	Expected Output
WN1	5	5	5	Equilateral
WN2	2	2	3	Isosceles
WN3	3	4	5	Scalene
WN4	4	1	2	Not a Triangle

Strong Robust Test cases

Test Case	a	b	c	Expected Output
SR1	-1	5	5	Value of a is not in the range of permitted values
SR2	5	-1	5	Value of b is not in the range of permitted values
SR3	5	5	-1	Value of c is not in the range of permitted values
SR4	-1	-1	5	Values of a, b are not in the range of permitted values
SR5	5	-1	-1	Values of b, c are not in the range of permitted values
SR6	-1	5	-1	Values of a, c are not in the range of permitted values
SR7	-1	-1	-1	Values of a, b, c are not in the range of permitted values

5 a) Write a short note on random testing

The basic idea is that, rather than always choose the min, min+, nom, max–, and max values of a bounded variable, use a random number generator to pick test case values. This avoids any form of bias in testing Tables 5.6 show the results of randomly generated test cases. They are derived from a Visual Basic application that picks values for a bounded variable:
$a \leq x \leq b$ as follows:
x = Int((b – a + 1) * Rnd + a)

where the function Int returns the integer part of a floating point number, and the function Rnd generates random numbers in the interval [0, 1]. The program keeps generating random test cases until at least one of each output occurs

Table 5.6 Random Test Cases for Triangle Program

Test Cases	*Nontriangles*	*Scalene*	*Isosceles*	*Equilateral*
1289	663	593	32	1
15,436	7696	7372	367	1
17,091	8556	8164	367	1
2603	1284	1252	66	1
6475	3197	3122	155	1
5978	2998	2850	129	1
9008	4447	4353	207	1
Percentage	49.83%	47.87%	2.29%	0.01%

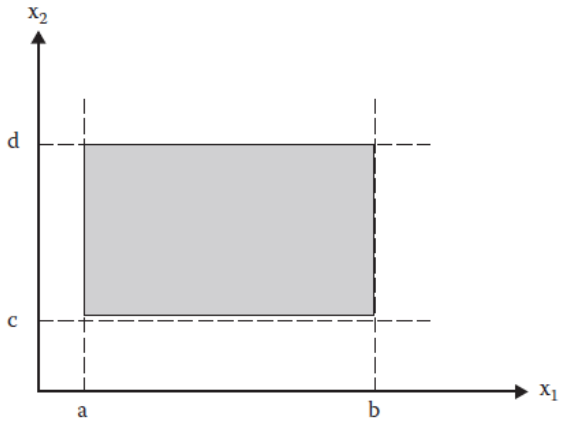
b) Discuss what typical test case information should include

A complete test case will contain a test case identifier, a brief statement of purpose (e.g., a business rule), a description of preconditions, the actual test case inputs, the expected outputs, a description of expected post conditions, and an execution history. The execution history is primarily for test management use—it may contain the date when the test was run, the person who ran it, the version on which it was run, and the pass/fail result

6) Justify the usage of boundary value analysis with function of two variables and highlight the limitations of BVA

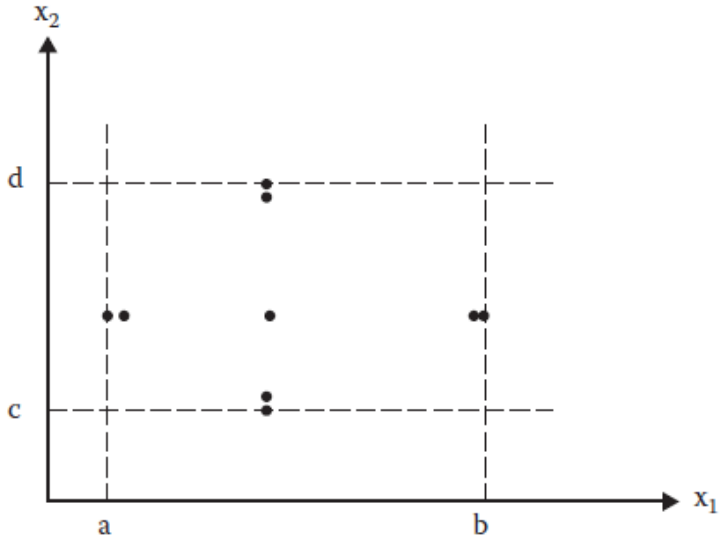
Consider function F, of two variables x1 and x2. When the function F is implemented as a program, the input variables x1 and x2 will have some (possibly unstated) boundaries:

$$a \leq x1 \leq b$$
$$c \leq x2 \leq d$$



Normal Boundary Value Testing: 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: min, min+, nom, max–, and max.

The normal 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>}

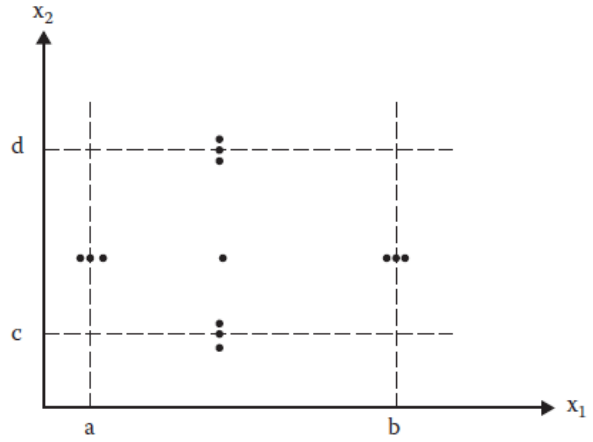


Thus, for a function of n variables, boundary value analysis yields 4n + 1 unique test cases.

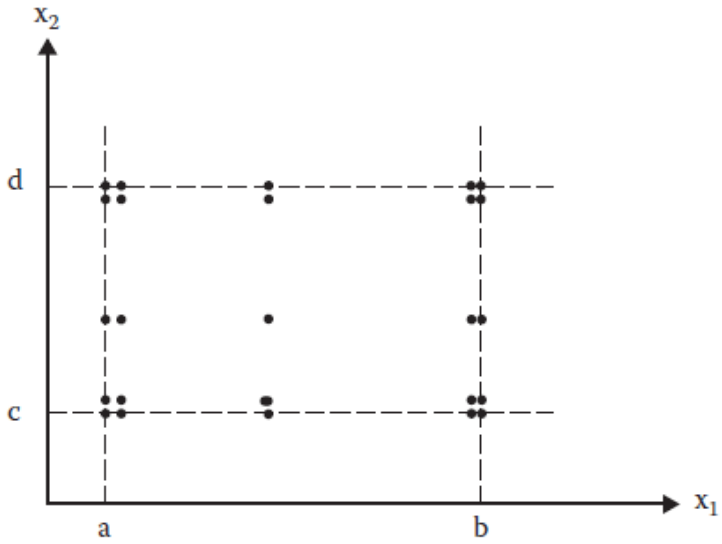
Robust Boundary Value Testing

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 : the extrema are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–)



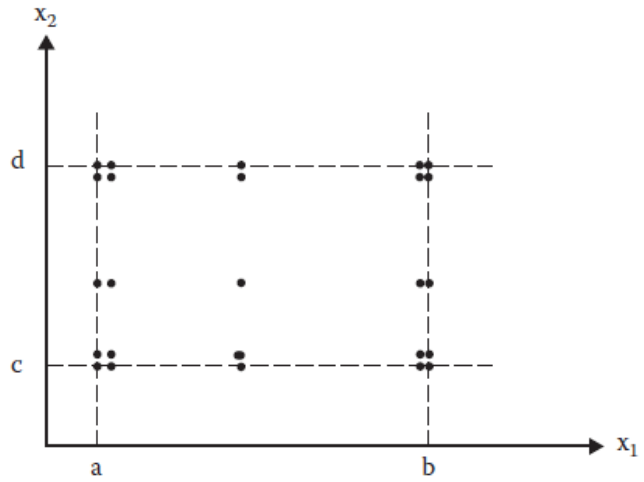
Worst-Case Boundary Value Testing: Rejects single-fault assumption and more than one variable has an extreme value. For each variable, we start with the five-element set that contains the min, min+, nom, max–, and max values. We then take the Cartesian product of these sets to generate test cases. The result of the two-variable version of this is shown in Figure



Worst-case boundary value testing is clearly more thorough in the sense that boundary value analysis test cases are a proper subset of worst-case test cases. It also represents much more effort:

worst-case testing for a function of n variables generates 5n test cases, as opposed to 4n + 1 test cases for boundary value analysis

Figure below shows the robust worst-case test cases for our two-variable function

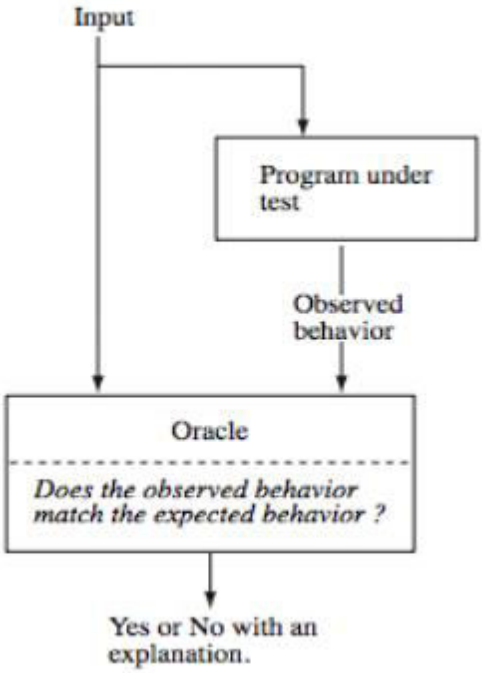


Limitations of Boundary Value Analysis

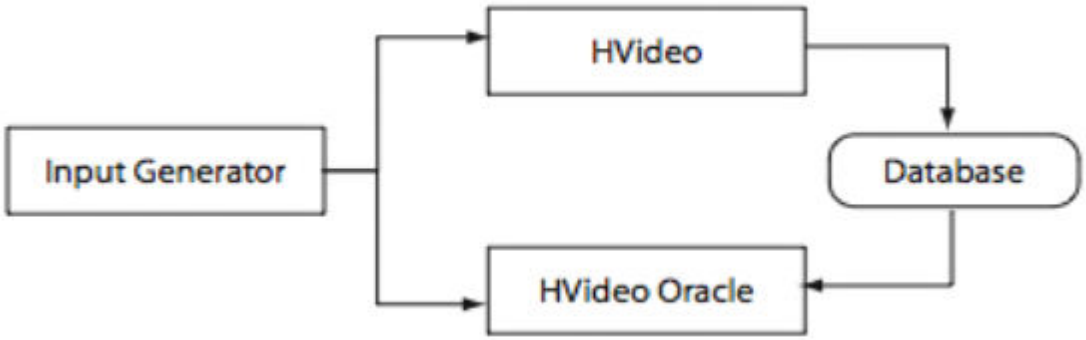
Boundary value analysis works well when the program to be tested is a function of several independent variables that represent bounded physical quantities Boundary value analysis test cases are derived from the extrema of bounded, independent variables that refer to physical quantities, with no consideration of the nature of the function, nor of the semantic meaning of the variables.

7 a) Define oracle and explain its construction with example

The entity that performs the task of checking the correctness of the observed behavior is known as an oracle.

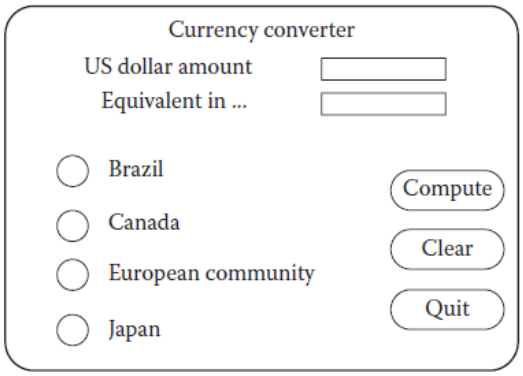


Oracle construction: Example



b) **Explain with diagram the currency converter.**

The currency conversion program is another event-driven program that emphasizes code associated with a GUI



The application converts US dollars to any of four currencies: Brazilian reals, Canadian dollars, European Union euros, and Japanese yen. Currency selection is governed by the radio buttons (option buttons), which are mutually exclusive. When a country is selected, the system responds by completing the label; for example, "Equivalent in …" becomes "Equivalent in Canadian dollars" if the Canada button is clicked. Also, a small Canadian flag appears next to the output position for the equivalent currency amount. Either before or after currency selection, the user inputs an amount in US dollars. Once both tasks are accomplished, the user can click on the Compute button, the Clear button, or the Quit button. Clicking on the Compute button results in the conversion of the US dollar amount to the equivalent amount in the selected currency. Clicking on the Clear button resets the currency selection, the US dollar amount, and the equivalent currency amount and the associated label. Clicking on the Quit button ends the application