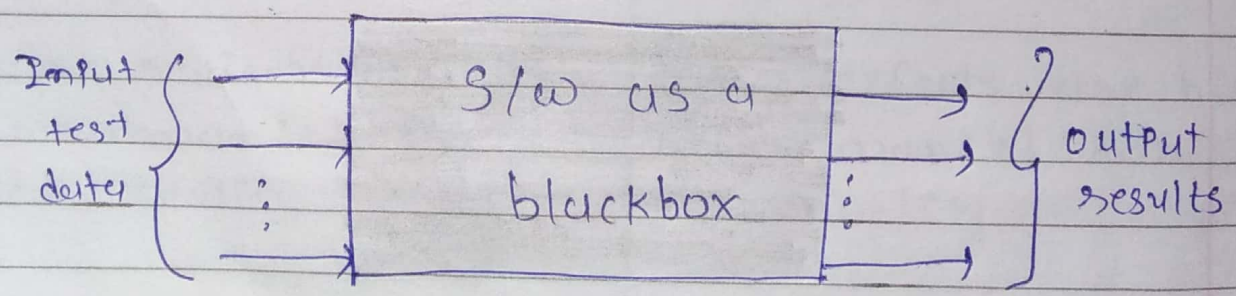
Name: Shah milani
Rollno: 92
Section: TY - B

ch: 1

Unit - 2 :- BlackBox Testing

* What is blackbox Testing :-

- It is a major technique in dynamic testing for designing effective testcases
- This technique considers only the functional requirements of the software or module In other words, the structural or logic of the software is not considered.
- Therefore this is also known as functional testing.
- The software system is considered as a blackbox, taking no notice of its internal structure, so it is also called as blackbox testing technique.



Blackbox Testing.

Scanned by CamScanner

- In blackbox testing, test cases are designed based on functional specifications. Input test data is given to the system, which is a blackbox to the tester, and results are checked against expected outputs after executing the software.
- ~~After executing~~

* Why blackbox-testing is Required :-

- blackbox testing attemps to find errors -
  (i) to test the modules independently
  (ii) to test the functional validity of the software so that incorrect or missing functions can be recognised.
  (iii) to look for interface errors
  (iv) to test the system behaviour and check its performance.
  (v) to test the maximum load or stress on the system.
  (vi) to test the software such that the client accepts the system within defined acceptable limits

Scanned by CamScanner

* When to do black-box testing :-

- blackbox testing activities require involvement of the testing team from the beginning of the software project life cycle.

- Testers can get involved right from the requirements gathering and analysis phase for the system under test. test scenarios and test data are prepared during the test construction phase of the STLC, when the software is in the design phase.

- once the code is ready and delivered for testing, test execution can be done. all the test scenarios develop earlier are executed.

Imp
* Blackbox testing techniques :-

(1) Boundary Value Analysis (BVA):

- An effective testcase design requires test cases to be designed such that they maximize the probability of finding errors.

Scanned by CamScanner

- Test cases designed with boundary input values have a high chance to find errors. It means that most of the failures crop up due to boundary values.

- BVA is a technique that uncovers the bugs at the boundary of input values. Here boundary means the maximum or minimum values taken by the input domain. for example, If A is an integer between 10-255, then boundary checking can be on 10 [9, 10, 11] and on 255 [256, 255, 254].

- Similarly, for B an integer between 10-100 boundary checking can be on 10 [9, 10, 11] and 100 [99, 100, 101].

- BVA offers several methods to design test cases as discussed in the following section:

(A) Boundary value checking [BVC]:

- In this method the test cases are design by holding one variable at its extreme value and other variables at their nomial values in the input domain

Scanned by CamScanner

- The variable at its extreme value can be selected at:
  (i) minimum value [min]
  (ii) value just about the minimum value [min+]
  (iii) maximum value [max]
  (iv) value just below the maximum value [max-]

- For two variables A and B the following test cases can be designed:

	A	B
1	Anom	Bmin
2	Anom	Bmin+
3	Anom	Bmax
4	Anom	Bmax-
5	Amin	Bnom
6	Amin+	Bnom
7	Amax	Bnom
8	Amax	Bnom
9	Anom	Bnom

- It can be generalized that for "n" variable in a module "4n+1" test cases can be designed with BVC method.

(B) Robustness Testing Method:

- The idea of BVC can be extended such that boundary values are exceded as:

Scanned by CamScanner

(i) A value just greater than the maximum value (max + 1)
(ii) A value just less the minimum value (max - 1)

- when test cases are design considering the above point in addition to BVC which is called Robustness testing.

- For the previous example (taken in BVC), be following test cases to the list of existing 9 test cases

10	Amin+	Bnom
11	Amin-	Bnom
12	Anom	Bmax+
13	Anom	Bmin-

- It can be generalized that for 'n' input variables in a module '6n+1' test cases can be design with robustness testing.

(c) Worst-case testing method:

- We can again extend the concept of BVC by assuming more than one variable on the boundary it is called worst case testing method.

Scanned by CamScanner

- Taking a same example of two variables A & B, we can add the following test cases to the existing 9 test cases of BVC

10	Amin	Bmin
11	Amin+	Bmin
12	Amax	Bmin
13	Amax-	Bmin
14	Amin	Bmin+
15	Amin+	Bmin+
16	Amax	Bmin+
17	Amax-	Bmin+
18	Amin	Bmax
19	Amin+	Bmax
20	Amax	Bmax
21	Amax-	Bmax
22	Amin	Bmax-
23	Amin+	Bmax-
24	Amax	Bmax-
25	Amax-	Bmax-

- It can be generalize that 'n' input variables in a module '6^n' testcases can be design with ~~works~~ ~~toorsk~~ worst-case testing

→ Conclusion:

- BVA is applicable when the module to be tested is a function of several independent

Scanned by CamScanner

variables. This method becomes important for physical quantity where boundary condition checking is crucial

- eg: Systems having requirements of minimum and maximum temperature pressure, speed etc. However it is not useful for boolean variable.

⟹ Example of Boundary value Analysis (BVC):

- A Program calculates A^B where A lies in the range [1,10] & B within [1,5] designed testcase for this program using BVC, robustness testing & worst-case testing method.

(A) Test cases using BVC:

- Since there are two variables 'A' & 'B' the total number of test cases will 4n+1 = 9. The set of boundary values is as follows:

	A	B
min value	1	1
mint value	2	2
max value	10	5
max- value	9	4
Nominal	5	3

Scanned by CamScanner

- Using this value test cases can be design show as below

Testcase id	A	B	expected output
1	1	3	1
2	2	3	8
3	10	3	1000
4	9	3	729
5	5	1	5
6	5	2	25
7	5	4	625
8	5	5	3125
9	5	3	125

(B) Test-cases using Robustness testing method:

- Since there are two variable 'A' & 'B'. The total no of test cases will "6n+1". The set of boundary value is as follows.

	A	B
min value	1	1
min- value	0	0
min+ "	2	2
max "	10	5
max+ "	11	6
max- "	9	4
Nominal "	5	3

Scanned by CamScanner

- using this values testcases can be design as shown below:

Testcase id	A	B	expected output
1	1	3	1
2	2	3	8
3	10	3	1000
4	9	3	729
5	5	1	5
6	5	2	25
7	5	4	625
8	5	5	3125
9	5	3	125
10	0	3	Invalid
11	11	3	Invalid
12	5	0	Invalid
13	5	6	Invalid

ee) Test-cases using worst case Testing:

- Since there are two variable a & b, The total numbers of testcases will be $5^n = 25$. The set of boundary values is shown below:

	A	B
min	1	1
min+	2	2
max	10	5
max-	9	4
Nominal	5	3

Scanned by CamScanner

- There may be more than one variable at extreme values in this case. Therefore testcases can be design as shown below:

Testcase id	A	B	expected o/p
1	1	1	1
2	1	2	1
3	1	3	1
4	1	4	1
5	1	5	1
6	2	1	2
7	2	2	4
8	2	3	8
9	2	4	16
10	2	5	32
11	5	1	5
12	5	2	25
13	5	3	125
14	5	4	625
15	5	5	3125
16	9	1	9
17	9	2	81
18	9	3	729
19	9	4	6561
20	9	5	59049
21	10	1	10
22	10	2	100
23	10	3	1000
24	10	4	10000
25	10	5	100000

Scanned by CamScanner

(2) Equivalance class Testing:

- we know that the input domain for testing is too large to test every input so we can divide or Partition the input domain based on a common feature or class of data.

- Equivalance Partitioning is a method for deriving testcases where in classes of input condition called 'equivalance classes' are Identified such that each member of the class causes the same kind of Processing and output to occur. Thus, instance of testing every input, only one testcase for in each Partitioned class can be executed.

- It means only one test case in the equivalance class will be sufficient to find the error. In this way testing covers the whole input domain. There by reducing the total number of testcases. equivalance partition method has the following:

(i) Completeness:

- without executing all the testcases we strive to touch the completeness of testing domain

Scanned by CamScanner

(ii) Non-redundancy:

- when the test cases are executed having inputs for the same then, their case redundancy in executing the testcases time & resources are wasted in executing this redundent testcases as they explore the same time of bug thus the goal of equivalance partining method is to reduce this redundant testcases.

- To use equivalance partitioning, one need to perform two cases:

(i) Identification of equivalent class:

- Different equivalance classes are formed by grouping inputs for which the behaviour pattern of the module is similar. For example, A module that finds the absolute value for integers specify different behaviour patterns for positive & negative integers so we will form two classes:

(a) Positive integers

(b) Negative integers

- Two types of classes can always be idenify:

(A) Valid Equivalance classes:

- This classes consider valid inputs to the program.

Scanned by CamScanner

(B)

Invalid Equivalance classes:

- one must not be restricted to valid input only we should also considered invalid inputs that will generate error condition or unexpected behaviour of the programming.

- Following are some guidelines for forming equivalance:

(a) If there is no reason to belive the entire range of an input will be treated in the same member, than the range should be split into two or more equivalance classes.

(b) If a program handles each valid input differently than define one valid equivalance class per valid input

(c) BVA can help a identifing the classes for example $a \leq a \leq 100$, one valid equivalance class can be form the valid range of A & two valid classes that is a greater than 100 can be identify

(d) If m input variable can identify more than one category than each category we can match equivalance classes.
eg, If a input was character than it can be

Scanned by CamScanner

an alphanet, a number or special character so we can make three valid classes for this input and one invalid class.

(e) If m input condition specify a must be "Situation". For eg, (First character of the input must be a letter) identify a valid equivalance class (It is ~~not~~ a letter) & invalid equivalance class (It is not a letter).

(f) Equivalance classes can be form according to the outputs desired in the Program

(g) Look for membership of m input condition in a set or group & identify valid (with in the set) and Invalid (Outside the set) classes

(ii) Identifing the test cases:

- (a) Assign a unique identification number to each equivalance class.

- (b) write a new test case covering as many of the uncovered valid equivalance classes as possible, untill all valid equivalance classes have been covered by test cases.

Scanned by CamScanner

- (c) write a test case that covers one of the uncovered invalid equivalance classes untill all invalid equivalance classes have been covered by test cases.

- eg, A program takes an angle as input within the range [0-360] & determines in which ~~quardant~~ quadrant the angle lies, design testcases using equivalance class partitioning methods.

- First we partition the domain of input as valid and invalid values getting the following classes

$I_1 = \{ \langle Angle \rangle : 0 \leq Angle \leq 360 \}$

$I_2 = \{ \langle Angle \rangle : Angle < 0 \}$

$I_3 = \{ \langle Angle \rangle : Angle > 360 \}$

- The test cases design from this classes are:

Testcase id	Angle	Expected Result	classes covered by the case
1	50	I ~~quardant~~ quadrant	I_1
2	-1	invalid	I_2
3	361	invalid	I_3

- The classes can also be prepared based on the output criteria:

$O_1 = \{ \langle Angle \rangle :$ First Quadrant, if $0 \leq Angle \leq 90 \}$

$O_2 = \{ \langle Angle \rangle :$ Second Quadrant, if $91 \leq Angle \leq 180 \}$

$O_3 = \{ \langle Angle \rangle :$ Third Quadrant, if $181 \leq Angle \leq 270 \}$

$O_4 = \{ \langle Angle \rangle :$ Fourth Quadrant, if $271 \leq Angle \leq 360 \}$

$O_5 = \{ \langle Angle \rangle :$ Invalid Angle, if $Angle < 0 \}$

$O_6 = \{ \langle Angle \rangle :$ Invalid Angle, if $Angle > 360 \}$

Scanned by CamScanner

- The testcases are designed from this classes are:

id	Angle	Expected Result	classes covered by cases
1	50	First Quadrant	O_1
2	135	second Quadrant	O_2
3	250	Third Quadrant	O_3
4	320	Fourth Quadrant	O_4
5	370	Invalid	O_5
6	-1	Invalid	O_6

- So after removing redundant test cases from input output criterias the above six testcases are sufficient for testing.

(3) <u>Decision Table Based Testing:</u>

- Boundary value analysis equilens class Partitioning.

- Decision table method do not consider each input seperately. There may be some critical behaviour to be tested when some combination of input condition are considered.

- Decision table method represents the information in a tabular method it can considered complex combination of input condition & resulting actions. Decision table use logical expression. each oprand or variable in a logical expression takes the value "TRUE" or "FALSE".

Scanned by CamScanner

- A decision is form with the following components:

		Rule 1	Rule 2	Rule 3	Rule 4
condition stufs	C1	True	True	False	1
	C2	False	True	False	True
	C3	True	True	True	1
action stufs	A1		X		
	A2	X			
	A3			X	

→ Condition Stufs:

It is a list of input condition for which the complex combination is made.

→ Action Stufs:

It is a list of resulting action which will be perform if a combination of input condition is satisfied.

→ Condition Entry:

It is a specific entry in the table corresponding to input conditions mentioned in the condition stuff.

- when we entered TRUE OR FALSE for all input condition for all particular combination then it is called a condition Entry. Thus a rule defines which combination of condition produces the resulting action when the condition entry takes only two values - True or False, then it is called "limited entry decision table."

Scanned by CamScanner

- when the condition entry takes several values, then it is called 'extended entry decision table'.

- when the condition entry takes several values, then it is called 'extended entry decision table'.

- In limited entry decision table, condition entry which has no effect, whether it is true or false < is called a 'don't care' state or 'immaterial state'.

- The state of don't care condition doesn't effect the resulting action.

→ Action Entry:

It is entry in the table for the resuting action to be perform when one rule is satisfied. 'x' denotes the action entry in the table.

- The guidlines to develop a decision table are as follows:

(i) List all action that can be associated with a specific module or procedure.

(ii) List all conditions during execution of the module.

Scanned by CamScanner

(iii) Associate specific sets of condition with specific action, eliminating impossible combination of condition, alternatively develop every possible permutation of condition.

(iv) Define rules by indicating what action occurs for a set of condition

→ Test case Design Using Decision Table:

(i) Interpret condition stubs as the inputs for the test case

(ii) Interpret action stubs as the expected output for the test case

(iii) Rule, which is the combination of input condition becomes the test case it self.

(iv) If there are 'k'-rules over 'n' binary conditions, there are atleast 'k' test cases and at the most 2^n test cases.

Scanned by CamScanner

→ Expanding the Immaterial cases in Decision Table:

- Immaterial cases (I) or don't care condition means that the value of a particular condition in the specific rule does not make a difference whether it is TRUE OR FALSE.

- However the don't care conditions should be throughly check. So, the rules in the table can be expanded. Sometimes expanding the decision table to spellout don't care conditions can reveal hidden problems.

- eg:

A wholesaler has three types of customers. Discount is given as per the following procedure:

(i) For DCS S&D orders, 10% disc given irrespective of the order value.

(ii) For orders of more than Rs 50,000, Agents get disc. 15% & Retailer get disc 10%.

(iii) For orders of less than Rs 20,000, ~~Agents~~ & more and upto Rs 50,000, Agents get 12% & Retailer get 8% disc.

Scanned by CamScanner

(iv) For orders of less than Rs 20,000, Agents get 8% and Retailer get 5% disc.

The above rules do not apply to the furniture items where 10% disc is given flat.

- Design testcases for this system using decision table testing.

Solution

	Entry							
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
Condition stub								
C1: DCS & D	T	F	F	F	F	F	F	1
C2: Agent	F	T	T	T	F	F	F	1
C3: Retailer	F	F	F	F	T	T	T	1
C4: Order > 50,000	1	T	F	F	T	F	F	1
C5: Order 20K-50K	1	F	T	F	F	T	F	1
C6: order < 20K	1	F	F	T	F	F	T	1
C7: Furniture	F	F	F	F	F	F	F	T
Action stub								
A1: 5% Disc							X	
A2: 8% Disc				X		X		
A3: 10% Disc	X				X			X
A4: 12% Disc			X					
A5: 15% Disc		X						

Scanned by CamScanner

- The testcase derived from the decision table are given below:

Testcase Id	Type of customer	Product Furniture?	Order Value (RS)	expected Result
1	DCRS&D	No	51,000	10% Disc
2	Agent	No	52,000	15% Disc
3	Agent	No	23,000	12% Disc
4	Agent	No	15,000	8% Disc
5	Retailer	No	53,000	10% Disc
6	Retailer	No	27,000	8% Disc
7	Retailer	No	18,000	5% Disc
8	Agent	YES	34,000	10% Disc

Scanned by CamScanner