For Updated version, please click on this
http://ocw.ump.edu.my

BCS3323 – Software Testing and Maintenance

Test Case Design
Black Box

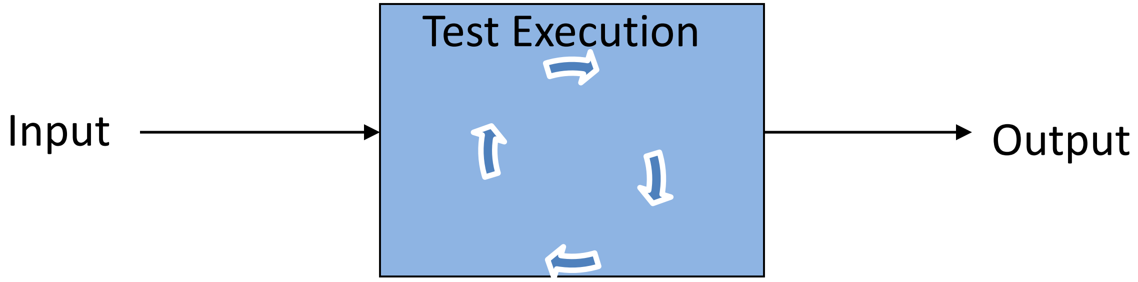
Editors
Dr. AbdulRahman A. Alsewari
Faculty of Computer Systems & Software Engineering
alswari@ump.edu.my

Black Box Testing

- **Black-box testing**: *Testing, either functional or non-functional, without reference to the internal structure of the component or system. (ISTQB)*

Black Box Testing

- testing without knowing the internal workings of the code
- WHAT a system does, rather than HOW it works it
- typically used at System Test phase, although can be useful throughout the test lifecycle
- also known as specification based testing and function testing
- Applies for Functional and Non-Functional testing



If Output = Expected result then pass

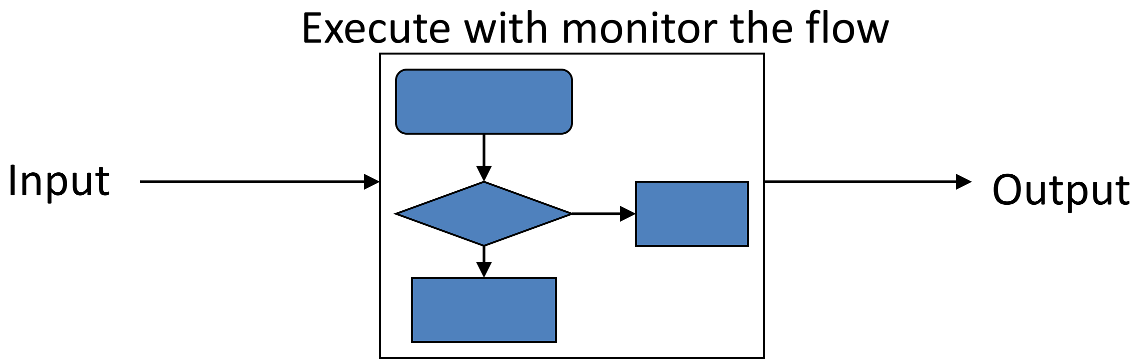
White Box Testing

white-box testing: *Testing based on an analysis of the internal structure of the component or system.*

(ISTQB)

White Box Testing

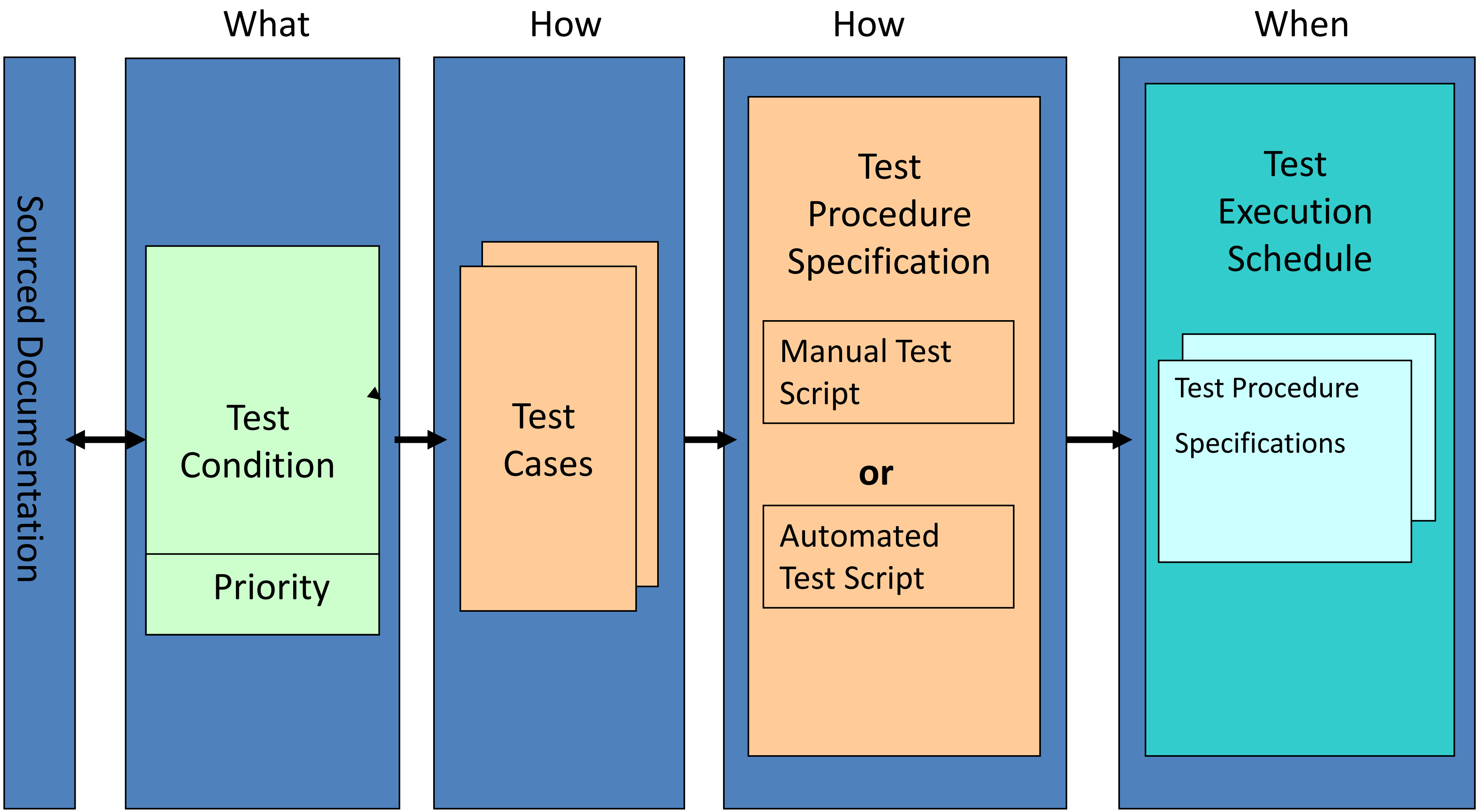
- testing based upon the structure of the code
- typically undertaken at Component and Component Integration Test phases by development teams
- also known as structural or glass box testing or structure based testing



Test Case Design

- Terminologies
  - Test condition – an item or event of a component or system that could be verified by one or more test cases
  - Test case specification– a set of input values, execution preconditions, expected results, and execution postconditions, designed for a specific test objective or test condition
  - A test procedure specification – a sequence of actions for the execution of the test. It may consist of number of test cases.
  - Test basis: All documents from which the SRS, SDS, code, or any related documents of a component or a system can be inferred.

Test Conditions, Cases, Procedures and Schedule



Illustrative Example of Test Case Design -1

In order for 3 integers a, b, and c to be the sides of a triangle, the following conditions must be met:

Scalene: a + b > c, where a<b<c

Isosceles: a + a > c, where b=a

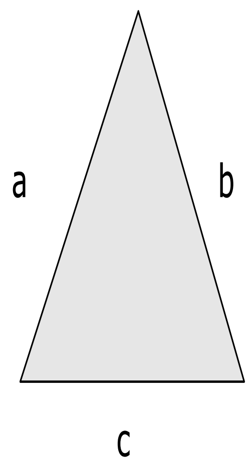
Equilateral: a = a = a, where b=a, c=a, and a >0

A triangle is:

Scalene if no two sides are equal

Isosceles if 2 sides are equal

Equilateral if all 3 sides are equal




```
void triangle (int a, int b, int c)
{
  int min,med, max;

  if (a>b)
   {
    max=a;
    min = b;
   }
  else
   {
    max = b;
    min = a;
   }
  if (c>max)
    max = c;
  else if (c<max)
    min = c;
  med = a+b+c-min-max;
  if (max>min+med)
    cout << "Impossible triangle\n";
  else if (max==min)
    cout << "Equilateral triangle\n";
  else if (max==med||med==min)
    cout << "Isoceles triangle\n";
  else if (max*max==min*min + med*med)
    cout << "Rightangled triangle\n";
  else
    cout << "Any triangle\n";
}
```

Illustrative Example of Test Case Design -2

Test Oracle – test cases with expected results

Conformance Analysis
- Test pass when expected = actual results

Conditions	Test ID	a	b	c	Expected Results	Actual Results
Scalene=> a + b > c, where a<b<c	#1	3	4	6	Scalene	
	#2	6	4	3	Scalene	
	#3	3	6	4	Scalene	
Isosceles=> a + a > c, where b=a	#4	2	2	3	Isosceles	
	#5	2	3	2	Isosceles	
	#6	3	2	2	Isosceles	
Equilateral => a = a = a, where b=a, c=a, and a >0	#7	4	4	4	Equilateral	
	#8	2	2	2	Equilateral	
Invalid Scalene => a+b ≤ c	#9	1	2	4	Not a triangle	
	#10	1	4	2	Not a triangle	
	#11	2	1	4	Not a triangle	
Invalid Isosceles => a+a ≤ c, where b=a	#12	2	2	4	Not a triangle	
	#13	2	4	2	Not a triangle	
	#14	4	2	2	Not a triangle	
Invalid input => a=0 or b=0 or c=0 or all a=b=c≤0	#15	0	1	2	Not a triangle	
	#16	2	0	1	Not a triangle	
	#17	1	2	0	Not a triangle	
	#18	0	0	0	Not a triangle	
	#19	5	-1	7	Not a triangle	
	#20	-5	-1	-1	Not a triangle	

Test Case

Test Suite – collection of test cases

Test Procedure

```
Procedure
1. Change directory to \UnitTest
2. Run project.Triangle
3. Enter all the values of a,b,c when prompted.
4. Check the printed output for actual value.
```

Why dynamic test techniques?

- Dynamic test technique is a sampling technique.
- Exhaustive testing is testing all potential inputs and conditions is unrealistic
  - So we need to use a subset of all potential test cases
  - Select the high likelihood of detecting defects
- There is a required processes to select the efficient and intelligent test cases
  - test case design techniques are such thought processes

What is a testing technique?

- a process for selecting or designing test cases based on a specification or structure model of the system
- successful when detecting defects
- 'best' practice
- It is a process of a best test cases derived
- a process of objectively evaluating the test effort

Testing should be rigorous, thorough and systematic

Advantages of techniques

- Different people: similar probability find faults
  - gain some independence of thought
- Effective testing: find more faults
  - focus attention on specific types of fault
  - know you're testing the right thing
- Efficient testing: find faults with less effort
  - avoid duplication
  - systematic techniques are measurable

Using techniques makes testing much more effective

Measurement

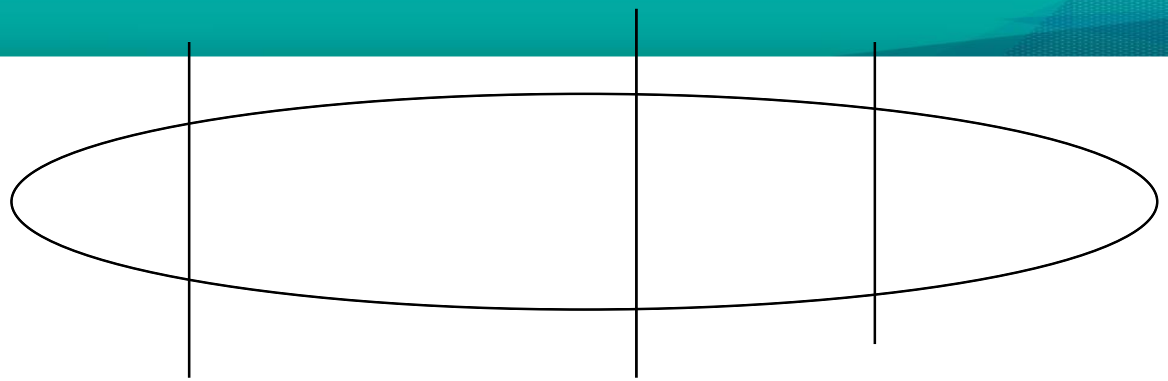
- Objective assessment of thoroughness of testing (with respect to use of each technique)
  - useful for comparison of one test effort to another
- E.g.

Project C
30% Boundaries partitions
40% Equivalence
70% Branches

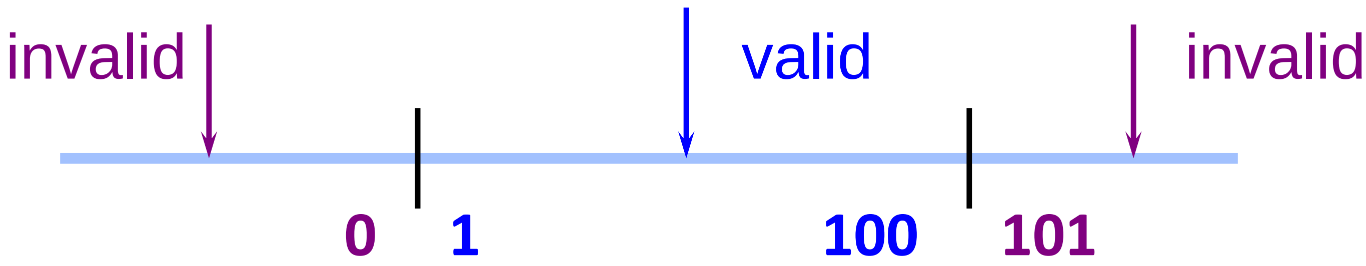
Project D
70% Branches partitions
50% Boundaries
45% Equivalence

Black Box Techniques for Test Case Design

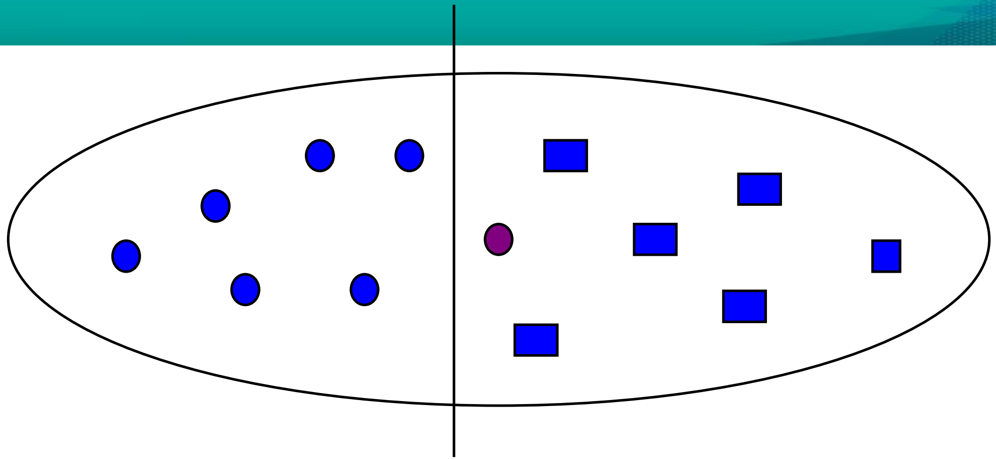
Equivalence partitioning (EP)



– primarily black box technique
– divide (partition) the inputs, outputs, etc. into areas which are the same (equivalent)
– assumption: if one value works, all will work
– one from each partition better than all from one



Boundary value analysis (BVA)

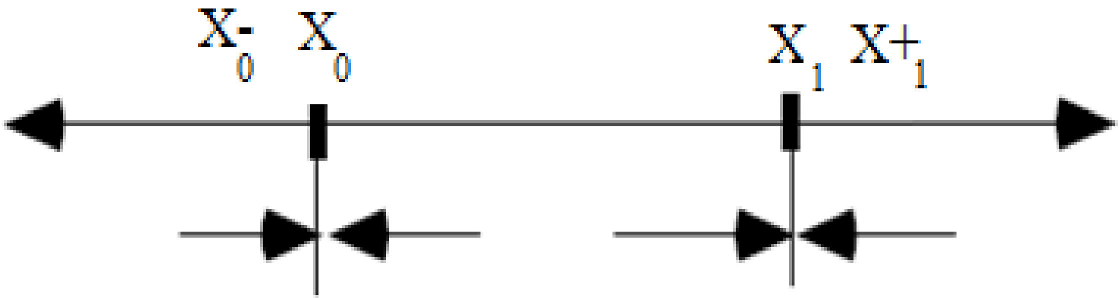


– faults tend to lurk near boundaries

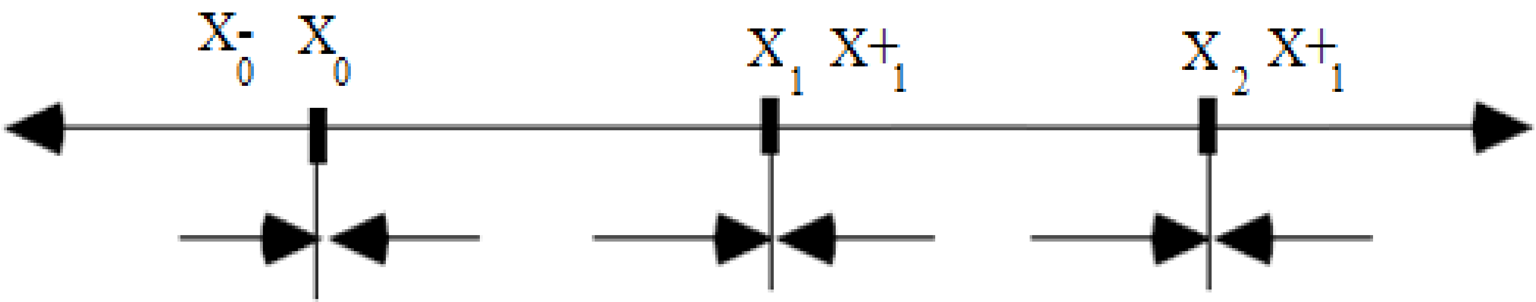
– good place to look for faults

– test values on both sides of boundaries

Boundary values are = $\{ X^-_0, X_0, X_1, X^+_1 \}$



Boundary values are = $\{ X^-_0, X_0, X_1, X^+_1, X_2, X^+_1 \}$



Rule of Thumb for BVA

Example 1: EP

```
void ValveControl (int pressure, int temperature)
 {
    if (pressure <= 10)
      {
        OpenTheValve();
        printf ("Valve opened\n");
      }
   if (pressure > 100)
     {
       CloseTheValve();
      printf ("Valve closed\n");
     }
   else
     {
       ShutDown();
     }

   if (temperature > 27)
     {
      EnableCoolingCoil();
      printf ("Cooling coil enabled\n");
     }
    else
    {
     DisableCollingCoil();
    }
 }
```

- Using EP and BV, derive the set of values for pressure and temperature.
- Enumerate exhaustively all the values of pressure and temperature to form a complete test suite.

Example 2: EP & BVA

- Scenario: If you take the train before 9:30 am or in the afternoon after 4:00pm until 7:30 pm (‘the rush hour’), you must pay full fare. A saver ticket is available for trains between 9:30 am and 4:00 pm.
  - Identify the partitions
  - Identify the boundary values to test train times for ticket type
  - Derive the test cases using EP and BVA

Example 3: EP

Consider a component, *generate_grading,* with the following specification:

The component is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark which is calculated as the sum of the exam and c/w marks, as follows:

greater than or equal to 70	-	*'A'*
greater than or equal to 50, but less than 70	-	*'B'*
greater than or equal to 30, but less than 50	-	*'C'*
less than 30	-	*'D'*

Where a mark is outside its expected range then a fault message ('FM') is generated. All inputs are passed as integers.

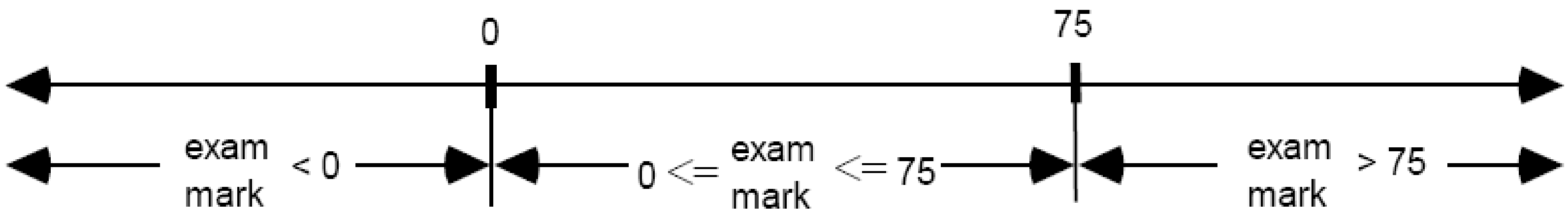
Valid partitions

- The valid partitions can be
  - 0<=exam mark <=75
  - 0<=coursework <=25

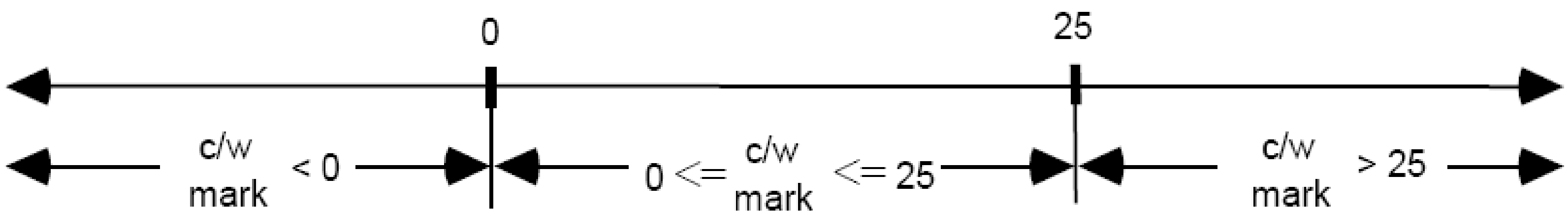
Invalid partitions

- The most obvious partitions are
  - Exam mark > 75
  - Exam mark < 0
  - Coursework mark > 25
  - Coursework mark <0

Exam mark and c/w mark



And for the input, coursework mark, we get:



Less obvious invalid input EP

- invalid INPUT EP should include

exam mark = real number (a number with a fractional part)
exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic

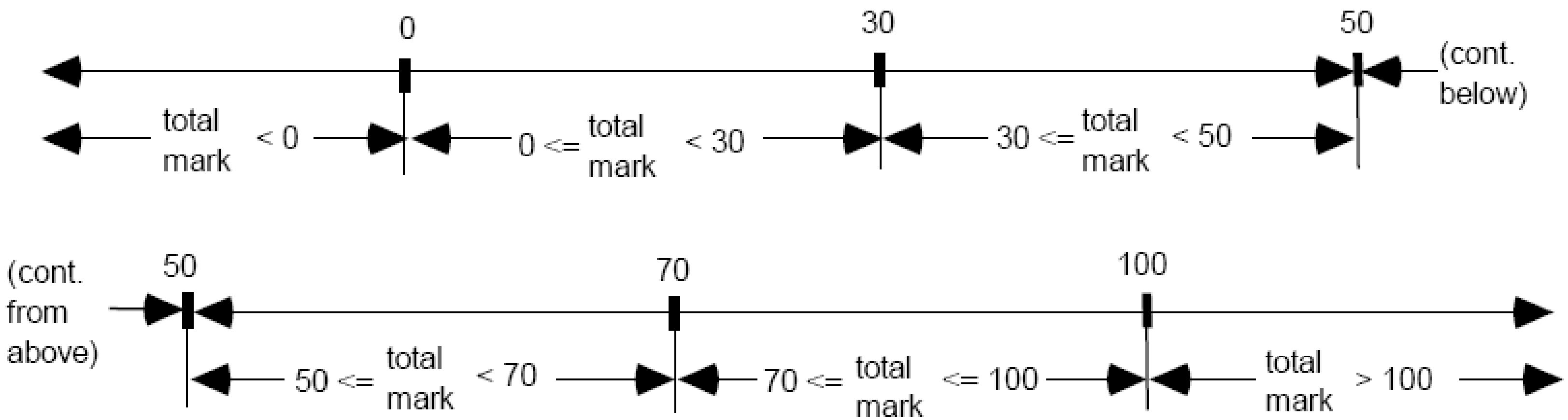
Partitions for the OUTPUTS

- EP for valid OUTPUTS should include

'A'	is induced by	70 <= total mark <= 100
'B'	is induced by	50 <= total mark < 70
'C'	is induced by	30 <= total mark < 50
'D'	is induced by	0 <= total mark < 30
'Fault Message'	is induced by	total mark > 100
'Fault Message'	is induced by	total mark < 0

The EP and boundaries

- The EP and boundaries for total mark



Unspecified Outputs

- Three unspecfied Outputs can be identified (very subjective)
  - Output = “E”
  - Output = “A+”
  - Output = “null”

Total EP

0 <= exam mark <= 75
exam mark > 75
exam mark < 0
0 <= coursework mark <= 25
coursework mark > 25
coursework mark < 0
exam mark = real number
exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic
70 <= total mark <= 100
50 <= total mark < 70
30 <= total mark < 50
0 <= total mark < 30
total mark > 100
total mark < 0
output = 'E'
output = 'A+'
output = 'null'

Test Cases corresponding to EP exam mark (INPUT)

Test Case	1	2	3
Input (exam mark)	44	-10	93
Input (c/w mark)	15	15	15
total mark (as calculated)	59	5	108
Partition tested (of exam mark)	0 <= e <= 75	e < 0	e > 75
Exp. Output	'B'	'FM'	'FM'

Test Case 4-6 (coursework)

Test Case	4	5	6
Input (exam mark)	40	40	40
Input (c/w mark)	8	-15	47
total mark (as calculated)	48	25	87
Partition tested (of c/w mark)	0 <= c <= 25	c < 0	c > 25
Exp. Output	'C'	'FM'	'FM'

Test case for Invalid inputs

The test cases corresponding to partitions derived from possible invalid inputs are:

Test Case	7	8	9	10
Input (exam mark)	48.7	q	40	40
Input (c/w mark)	15	15	12.76	g
total mark (as calculated)	63.7	not applicable	52.76	not applicable
Partition tested	exam mark = real number	exam mark = alphabetic	c/w mark = real number	c/w mark = alphabetic
Exp. Output	'FM'	'FM'	'FM'	'FM'

Test cases for outputs:1

The test cases corresponding to partitions derived from the valid outputs are:

Test Case	11	12	13
Input (exam mark)	-10	12	32
Input (c/w mark)	-10	5	13
total mark (as calculated)	-20	17	45
Partition tested (of total mark)	t < 0	0 <= t < 30	30 <= t < 50
Exp. Output	'FM'	'D'	'C'

Test cases for outputs:2

Test Case	14	15	16
Input (exam mark)	44	60	80
Input (c/w mark)	22	20	30
total mark (as calculated)	66	80	110
Partition tested (of total mark)	50 <= t < 70	70 <= t <= 100	t > 100
Exp. Output	'B'	'A'	'FM'

Test cases for invalid outputs:3

The test cases corresponding to partitions derived from the invalid outputs are:

Test Case	17	18	19
Input (exam mark)	-10	100	null
Input (c/w mark)	0	10	null
total mark (as calculated)	-10	110	null+null
Partition tested (output)	'E'	'A+'	'null'
Exp. Output	'FM'	'FM'	'FM'