Black Box Testing: Equivalence Partitioning

Introduction

Equivalence partitioning is based on the premise that the inputs and outputs of a component can be partitioned into classes that, according to the component's specification, will be treated similarly by the component. Thus the result of testing a single value from an equivalence partition is considered representative of the complete partition. **Example**

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.*

Initially the equivalence partitions are identified and then test cases derived to exercise the partitions. Equivalence partitions are identified from both the inputs and outputs of the component and both valid and invalid inputs and outputs are considered.

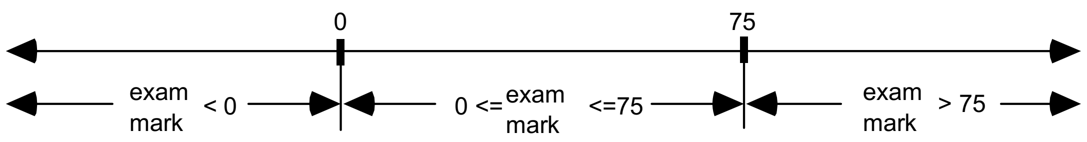
The partitions for the two inputs are initially identified. The *valid* partitions can be described by:

$0 \leq$ exam mark ≤ 75
$0 \leq$ coursework mark ≤ 25

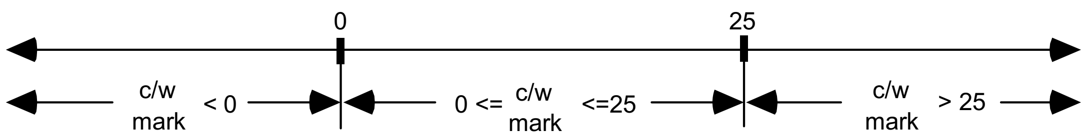
The most obvious *invalid* partitions based on the inputs can be described by:

exam mark > 75 exam
mark < 0 coursework
mark > 25
coursework mark < 0

Partitioned ranges of values can be represented pictorially, therefore, for the input, exam mark, we get:



And for the input, coursework mark, we get:



Less obvious invalid input equivalence partitions would include any other inputs that can occur not so far included in a partition, for instance, non-integer inputs or perhaps non-numeric inputs. So, we could generate the following invalid input equivalence partitions:

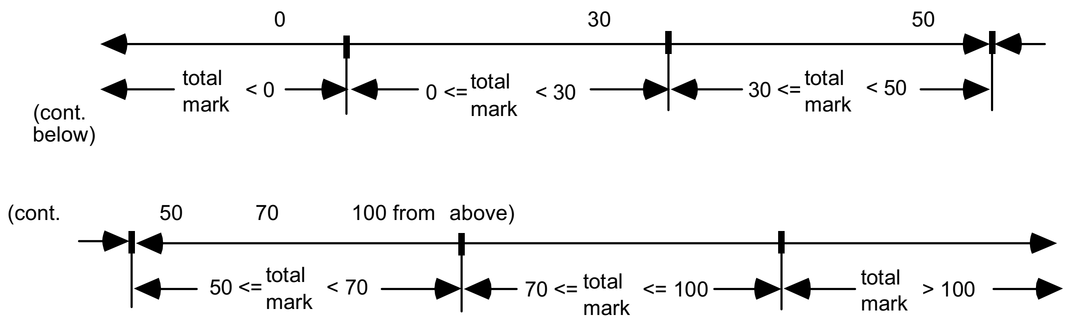
exam mark = real number (a number with a fractional part)
exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic

Next, the partitions for the outputs are identified. The *valid* partitions are produced by considering each of the valid outputs for the component:

'A'	is induced by	$70 \leq$ total mark ≤ 100
'B'	is induced by	$50 \leq$ total mark < 70
'C'	is induced by	$30 \leq$ total mark < 50
'D'	is induced by	$0 \leq$ total mark < 30
'Fault Message'	is induced by	total mark > 100
'Fault Message'	is induced by	total mark < 0

where total mark = exam mark + coursework mark. Note that 'Fault Message' is considered as a valid output as it is a *specified* output.

The equivalence partitions and boundaries for total mark are shown pictorially below:



An invalid output would be any output from the component other than one of the five specified. It is difficult to identify unspecified outputs, but obviously they must be considered as if we can cause one then we have identified a flaw with either the component, its specification, or both. For this example three unspecified outputs were identified and are shown below. This aspect of equivalence partitioning is very subjective and different testers will inevitably identify different partitions which *they* feel could possibly occur.

output = 'E'
output = 'A+'
output = 'null'

Thus the following nineteen equivalence partitions have been identified for the component (remembering that for some of these partitions a certain degree of subjective choice was required, and so a different tester would not necessarily duplicate this list exactly):

$0 \leq$ exam mark ≤ 75
exam mark > 75
exam mark < 0
$0 \leq$ coursework mark ≤ 25
coursework mark > 25
coursework mark < 0
exam mark = real number

exam mark = alphabetic
coursework mark = real number
coursework mark = alphabetic
$70 \leq$ total mark ≤ 100
$50 \leq$ total mark < 70
$30 \leq$ total mark < 50
$0 \leq$ total mark < 30
total mark > 100
total mark < 0
output = 'E'
output = 'A+'
output = 'null'

Having identified all the partitions then test cases are derived that 'hit' each of them. Two distinct approaches can be taken when generating the test cases. In the first a test case is generated for each identified partition on a one-to-one basis, while in the second a minimal set of test cases is generated that cover all the identified partitions.

The one-to-one approach will be demonstrated first as it can make it easier to see the link between partitions and test cases. For each of these test cases only the single partition being targetted is stated explicitly. Nineteen partitions were identified leading to nineteen test cases.

The test cases corresponding to partitions derived from the input exam mark are:

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)	$0 \leq e \leq 75$	$e < 0$	$e > 75$
Exp. Output	'B'	'FM'	'FM'

Note that the input coursework (c/w) mark has been set to an arbitrary valid value of 15.

The test cases corresponding to partitions derived from the input coursework mark are:

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)	$0 \leq c \leq 25$	$c < 0$	$c > 25$
Exp. Output	'C'	'FM'	'FM'

Note that the input exam mark has been set to an arbitrary valid value of 40.

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'

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)	$t < 0$	$0 \leq t < 30$	$30 \leq t < 50$
Exp. Output	'FM'	'D'	'C'

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)	$50 \leq t < 70$	$70 \leq t \leq 100$	$t > 100$
Exp. Output	'B'	'A'	'FM'

The input values of exam & coursework marks have been derived from the total mark, which is their sum.

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'

It should be noted that where invalid input values are used (as above, in test cases 2, 3, 5-11, and 1619) it may, depending on the implementation, be impossible to actually execute the test case. For instance, in Ada, if the input variable is declared as a positive integer then it will not be possible to assign a negative value to it. Despite this, it is still worthwhile *considering* all the test cases for completeness.

It can be seen above that several of the test cases are similar, such as test cases 1 and 14, where the main difference between them is the partition targetted. As the component has two inputs and one output, each test case actually 'hits' three partitions; two input partitions and one output partition.. Thus it is possible to generate a smaller 'minimal' test set that still 'hits' all the identified partitions by deriving test cases that are designed to exercise more than one partition. The following test case suite of eleven test cases corresponds to the minimised test case suite approach where each test case is designed to hit as many new partitions as possible rather than just one. Note that here all three partitions are explicitly identified for each test case.

Test Case	1	2	3	4
Input (exam mark)	60	40	25	15
Input (c/w mark)	20	15	10	8
total mark (as calculated)	80	55	35	23
Partition (of exam mark)	$0 \le e \le 75$	$0 \le e \le 75$	$0 \le e \le 75$	$0 \le e \le 75$
Partition (of c/w mark)	$0 \le c \le 25$	$0 \le c \le 25$	$0 \le c \le 25$	$0 \le c \le 25$
Partition (of total mark)	$70 \le t \le 100$	$50 \le t < 70$	$30 \le t < 50$	$0 \le t < 30$
Exp. Output	'A'	'B'	'C'	'D'

Test Case	5	6	7	8
Input (exam mark)	-10	93	60.5	q
Input (c/w mark)	-15	35	20.23	g
total mark (as calculated)	-25	128	80.73	-
Partition (of exam mark)	$e < 0$	$e > 75$	e = real number	e = alphabetic
Partition (of c/w mark)	$c < 0$	$c > 25$	c = real number	c = alphabetic
Partition (of total mark)	$t < 0$	$t > 100$	$70 \le t \le 100$	-
Exp. Output	'FM'	'FM'	'FM'	'FM'

Test Case	9	10	11
Input (exam mark)	-10	100	'null'
Input (c/w mark)	0	10	'null'
total mark (as calculated)	-10	110	null+null
Partition (of exam mark)	$e < 0$	$e > 75$	-
Partition (of c/w mark)	$0 \leq c \leq 25$	$0 \leq c \leq 25$	-
Partition (of total mark)	$t < 0$	$t > 100$	-
Partition (of output)	'E'	'A+'	'null'
Exp. Output	'FM'	'FM'	'FM'

The one-to-one and minimised approaches represent the two approaches to equivalence partitioning. The disadvantage of the one-to-one approach is that it requires more test cases and if this causes problems a more minimalist approach can be used. Normally, however, the identification of partitions is far more time consuming than the generation and execution of test cases themselves and so any savings made by reducing the size of the test case suite are relatively small compared with the overall cost of applying the technique. The disadvantage of the minimalist approach is that in the event of a test failure it can be difficult to identify the cause due to several new partitions being exercised at once. This is a debugging problem rather than a testing problem, but there is no reason to make debugging more difficult than it is already.

Some testers would say that a variable's input domain equivalence partitions must be combined with every other variables' input domains equivalence partitions. This is an extreme view as it leads to an explosion of number of tests. It hard however to argue against it on the ground of completeness.