
Software Quality Assurance: SOFT3302

Supplementary Lecture Material – Week 3

Introduction

This document provides an example of equivalence partitioning and boundary value analysis.

Background

A text processing facility used by an health department analyses free text and replaces various descriptions of oxygen saturation (SaO_2), which may be written in many different way, with a single canonical notation. There is a function that takes the various components derived from an inspection of text and calculates the canonical description.

The inputs to the function consist of an integer percentage value *reading* and a string containing a word indicating breathing might be assisted. No assistance is indicated by an empty word. Words indicating assistance come from the list MASK, NRB, NRBM, NEB, NEBULISER, NP, PRONGS and O2. Other words are unrecognised.

For legal input, the function must produces a result of the form O2STATUS= n where n is an integer as follows. (In these examples for the sake of brevity we shall refer to n rather than O2STATUS= n .)

Unassisted Breathing		
Reading	Result	Part. ID
$98 \leq r \leq 100$	1	OU1
$93 \leq r < 98$	2	OU2
$89 \leq r < 93$	3	OU3
$0 \leq r < 89$	4	OU4

Assisted Breathing		
Reading	Result	Part. ID
$93 \leq r \leq 100$	5	OA1
$89 \leq r < 93$	6	OA2
$0 \leq r < 89$	7	OA3

Other legal input should produce a result of -1 (which we shall term output partition OI). Illegal input (if it is possible to enter such) should raise an exception. (These partition identifiers will be used in the tables later for brevity.)

Equivalence Partitions

Inputs

The input reading has three partitions: $r < 0$, $0 \leq r \leq 100$, $r > 100$, which we shall refer to as IR1, IR2 and IR3 respectively.

The input assistance word has three partitions: the empty word (indicating no assistance), the set of words {MASK, NRB, NRBM, NEB, NEBULISER, NP, PRONGS, O2} indicating assistance and everything else. We shall refer to these as IA1, IA2 and IA3 respectively.

This gives rise to six valid input partition tests ($3 + 3$) if we are performing one-to-one testing.

Input reading valid partitions			
Test case	1	2	3
Input (reading)	-5	90	120
Input (assistance)	“	“	“
Partition tested (of reading)	$r < 0$ (IR1)	$0 \leq r \leq 100$ (IR2)	$r > 100$ (IR3)
Expected output	-1	3	-1

And

Input assistance valid partitions			
Test case	4	5	6
Input (reading)	90	90	90
Input (assistance)	“	‘MASK’	‘TEAPOT’
Partition tested (of assistance)	<i>No assistance</i> (OA1)	<i>Assistance</i> (OA2)	<i>Unrecognised</i> (OA3)
Expected output	3	6	-1

If this was taking place in a weakly-typed language where it might be possible to pass an argument of an incorrect type, or, say, input was *via* an interface where a user could enter a value of the incorrect type and/or format it would be necessary to include invalid value test cases as well. Indeed, if the behaviour for integer input readings outside the range 0 to 100 had not been specified these would need to also be included as invalid input tests. The expected outcome, *e.g.* exception, would need to be documented.

Input reading and assistance invalid partitions/values			
Test case	7	8	9
Input (reading)	‘ten’	80.6	90
Input (assistance)	“	‘MASK’	12
Partition tested (of reading)	<i>String</i>	<i>Real</i>	
Partition tested (of assistance)			<i>Non-string</i>
Expected outcome	<i>exception</i>	<i>exception</i>	<i>exception</i>

Notice that inputs are not tested according to output partitioning. These form separate output partitioning tests. Each input is also tested in isolation.

This technique may be varied by forming the cross-product of the various input partitions rather than just the addition. That is, for each partition of reading vary the assistance partition. This gives rise to $3 \times 3 = 9$ tests of valid inputs rather than the 6 tests shown above. This could be used in the case that there is deemed to be a risk of the various combinations of inputs affecting the result adversely.

Another variation is to choose different input values for the input not being tested from valid partitions. The advantage is that slightly more coverage of the input space is gained. A disadvantage (and this is a disadvantage of the ensuing debugging process and not of testing itself) is that it may make it slightly more difficult to trace a problem in the case of a failed test.

Outputs

There are eight output partitions as shown in the above tables and the specification for illegal input indicator (4 + 3 + 1). For each output partition, inputs need to be chosen to cause the necessary output. There may be multiple ways to achieve the desired output value a single one is chosen for each output.

Non-assisted breathing output partitions				
Test case	10	11	12	13
Input (reading)	99	95	91	50
Input (assistance)	‘	‘	‘	‘
Partition tested (of output)	$98 \leq r \leq 100$ (OU1)	$93 \leq r < 98$ (OU2)	$89 \leq r < 93$ (OU3)	$0 \leq r < 89$ (OU4)
Expected result	1	2	3	4

Assisted breathing output partitions			
Test case	14	15	16
Input (reading)	95	91	50
Input (assistance)	‘MASK’	‘MASK’	‘MASK’
Partition tested (of output)	$93 \leq r \leq 100$ (OA1)	$89 \leq r < 93$ (OA2)	$0 \leq r < 89$ (OA3)
Expected result	5	6	7

Other output partitions	
Test case	17
Input (reading)	-10
Input (assistance)	‘MASK’
Partition tested (of output)	<i>Illegal input indicator</i> (OI)
Expected result	-1

Notice that this results in 17 tests for this single method!

Minimal Testing

Some of these test cases test multiple partition. For example, test #5 tests the reading input partition $0 \leq r \leq 100$ as per test #2 as well as the output partition tested by test #15. A minimal set of tests could be designed which tested multiple partitions at once and which therefore used fewer test cases.

Minimal test cases covering all valid input equivalence classes.									
Test case	M1	M2	M3	M4	M5	M6	M7	M8	M9
Input (reading)	-10	99	110	95	91	50	95	91	50
Input (assistance)	“	‘NEB’	‘X’	“	“	“	“	‘NEB’	‘NEB’
Partition tested (of reading)	IR1	IR2	IR3	IR2	IR2	IR2	IR2	IR2	IR2
Partition tested (of assistance)	IA1	IA2	IA3	IA1	IA1	IA1	IA1	IA2	IA2
Partition tested (of output)	OI	OA1	OI	OUI	OUI	OUI	OUI	OA2	OA3
Expected result	-1	5	-1	1	2	3	4	6	7

This gives 9 test cases in total. Here test case #M2 is used to test the three classes IR2, IA2 and OA1.

Boundary Values

Boundary value testing involves testing the boundaries of equivalence classes. Values are tested on either side of the boundary. Testing inequalities requires at least 2 test points comprising one ‘in’ value: a point inside the equivalence class and one ‘out’ value). These two points should be one ‘on’ point and one ‘off’ point: a value on the boundary and a value off the boundary. For relational operators involving equality which specify closed boundaries to a region, *e.g.* \leq , the ‘on’ point is an ‘in’ value so the ‘off’ point should be an ‘out’ value. For relational operators such as $>$ which specify open boundaries, the ‘on’ point is an ‘out’ value so the ‘off’ point should be an ‘in’ value.

Strict equality (and maximal testing of inequalities) of ordered values requires 3 test points: one ‘on’ point and two ‘off’ points. These points shall be 1 ‘in’ and 2 ‘out’ values for equalities and 2 ‘in’ and one ‘out’ value for relational comparisons.¹

Testing of unordered quantities, such as strings, requires one ‘in’ and out ‘out’ value.

‘Out’ values should differ from ‘in’ values by the smallest amount possible.

Input reading boundary tests						
Test case	1	2	3	4	5	6
Input (reading)	-1	0	1	99	100	101
Input (assistance)	“	“	“	“	“	“
Boundary tested	0			100		
Expected output	-1	4	4	1	1	-1

¹It may be helpful to draw pictures if you find this unclear. See *Binder*, pp. 405ff.

Input assistance 'boundary' tests				
Test case	7	8	9	10
Input (reading)	99	99	99	99
Input (assistance)	"	'X'	'NEB'	'NEC'
Boundary tested	"		<i>Assistance</i>	
Expected output	1	-1	5	-1

Non-assisted breathing output boundary tests															
Test case	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Input (reading)	99	100	101	97	98	99	92	93	94	88	89	90	-1	0	1
Input (assistance)	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"
Boundary tested	100			98			93			89			0		
Expected output	1	1	-1	2	1	1	3	2	2	4	3	3	-1	4	4

Assisted breathing output boundary tests												
Test case	26	27	28	29	30	31	32	33	34	35	36	37
Input (reading)	99	100	101	97	98	99	92	93	94	88	89	90
Input (assistance)	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'	'O2'
Boundary tested	100			93			89			0		
Expected output	5	5	-1	6	5	5	7	6	6	-1	7	7

This is 37 tests! Have you been doing enough testing?

Fortunately all of these tests require only two harnesses: one to check a result; and one to check for an exception. The test suite could be table-driven. Working out the test cases may require more effort but preparing the actual test code should not be too onerous.

Implementation limits on data types

It may also be necessary to test at the limits of various data types realising that $x < 0$ may actually mean $-32768 \leq x < 0$ if x is a 2-byte signed integer. For example -32768 and 32767 for 2-byte signed integers or 0 and 65535 for 2-byte unsigned integers. Similarly with -2147483648 and 2147483647 or 0 and 4294967295 for 4-byte integers.