THEORICAL WORK 2 SOFTWARE ENGENEERING II

GROUP: A03 ISO2 – 2021-22

PROBLEM 3

Andrés Castellanos Cantos

Pablo Valverde Soriano

0. STATEMENT OF THE PROBLEM

A certification company grants certificates of the quality level of software products if a series of conditions are fulfilled. The aim is to develop and test a program that determines whether a software product can be certifiable, and if so, what level of certification it would obtain.

Only two quality characteristics will be considered: Functional Suitability and Maintainability. For this purpose, the indications in the following tables will be considered.

For functional suitability, the following table has been declared:

Measurement range	Functional Completeness	Functional Correctenes	Functional Appropriateness
[0,10)	0	0	0
[10, 35)	1	1	2
[35,50)	2	1	2
[50, 70)	2	2	3
[70, 90)	3	3	4
[90, 100]	4	5	5

The way to interpret this table is to consider a function of minima:

Functional Suitability = Min {Functional Completeness, Functional Correctness, Functional Appropriateness}

For example, functional completeness has been measured with a level of 55, functional correctness with a level of 86, and functional appropriateness with a level of 19. Looking at the tables, for functional completeness an equivalent value of 2 is obtained, for functional correctness it would be 3, and for functional Appropriateness 1.

Consequently, Functional Suitability can be calculated as Min {2,3,1} = 1.

For maintainability, there would be a similar matrix with each of the corresponding characteristics.

Maintainability = Min {Modularity, Reusability, Analyzability, Ability to be modified, Ability to be tested}

Rango Mediciones	Modularity	Reusability	Analyzability	Ability to be modified	Ability to be tested
[0,10)	0	0	0	0	0
[10, 35)	1	1	0	1	1
[35,50)	2	2	1	2	1
[50, 70)	2	2	2	3	2
[70, 90)	3	3	3	4	4
[90, 100]	4	5	5	5	4

Finally, the following table is used to calculate the overall quality level of the software product (it can only be certified when a level 3 is obtained):

		Mantenibilidad				
		1	2	3	4	5
Adecuación Funcional	1	1	1	1	1	1
	2	1	2	2	2	2
	3	2	2	3	3	3
	4	3	3	3	3	4
	5	3	3	4	4	5

If for some reason, some of the base measurements for the subcharacteristics could not be obtained, the corresponding exception will be thrown (there is no need to create exceptions for each type of subcharacteristics).

1. CODE OF THE PROBLEM

```
import java.io.IOException;
import java.util.stream.IntStream;

public class Ej3 {
	static int [][] FUNCTIONAL_MATRIX = {{0,0,0},{1,1,2},{2,1,2},{2,2,3},{3,3,4},{4,5,5}};
	static int [][] MAINTAINABILITY_MATRIX =
{{0,0,0,0,0},{1,1,0,1,1},{2,2,1,2,1},{2,2,2,3,2},{3,3,3,4,4},{4,5,5,5,4}};
	static int [][] ADECUATION_FUNCTIONAL_MATRIX =
{{1,1,1,1,1},{1,2,2,2,2},{2,2,3,3,3},{3,3,3,3,4},{3,3,4,4,5}};
	static int [] functional_input = {0,0,0};
	static int [] maintainability_input = {0,0,0,0,0};
	public static void main(String[] args) {
		try {
			int completeness_value = get_Value (FUNCTIONAL_MATRIX, functional_input[0], 0);
			int correcteness_value = get_Value (FUNCTIONAL_MATRIX, functional_input[1], 1);
			int appropiateness_value = get_Value (FUNCTIONAL_MATRIX, functional_input[2],
2);
			int min_functional = calculate_min_functional (completeness_value,
correcteness_value, appropiateness_value);
			int modularity_value = get_Value (MAINTAINABILITY_MATRIX,
maintainability_input[0], 0);
			int reusability_value = get_Value (MAINTAINABILITY_MATRIX,
maintainability_input[1], 1);
			int analyzability_value = get_Value (MAINTAINABILITY_MATRIX,
maintainability_input[2], 2);
			int modified_value = get_Value (MAINTAINABILITY_MATRIX,
maintainability_input[3], 3);
			int tested_value = get_Value (MAINTAINABILITY_MATRIX, maintainability_input[4],
4);
			int min_maintainability = calculate_min_maintainability (modularity_value,
reusability_value, analyzability_value, modified_value, tested_value);
			int value_adecuational_functional = ADECUATION_FUNCTIONAL_MATRIX
[min_functional][min_maintainability];
			System.out.println(value_adecuational_functional);
		}catch(Exception e){
			System.out.println(e);
		}
	}

	public static int calculate_min_functional (int completeness_value, int correcteness_value,
int appropiateness_value) {
		return Math.min(Math.min(correcteness_value, appropiateness_value),
appropiateness_value);
	}
```

```
	public static int calculate_min_maintainability (int modularity_value, int
reusability_value, int analyzability_value, int modified_value, int tested_value) {
		return IntStream.of(modularity_value, reusability_value,
analyzability_value, modified_value, tested_value).min().getAsInt();
	}

	public static int get_Value (int [][] matrix, int input, int type) throws
IOException {
		int range = get_range(input);
		int value = matrix[range][type];
		return value;
	}

	public static int get_range (int input) throws IOException {
		int range = 0;
		if (input < 0 || input > 100 ) {
			throw new IOException ();
		} else if (input < 10) {
			range = 0;
		} else if (input < 35) {
			range = 1;
		} else if (input < 50) {
			range = 2;
		} else if (input < 70) {
			range = 3;
		} else if (input < 90) {
			range = 4;
		} else {
			range = 5;
		}
		return range;
	}
}
```

2. VARIABLES THAT MUST BE CONSIDERED TO TEST THE METHOD

— **functional_input**: is an integer vector.

— **maintainability_input**: integer vector

— **completeness_value**, **correctness_value**, **appropriateness_value**: inputs to calculate the functional suitability.

— **modularity_value**, **reusability_value**, **analyzability_value**, **modified_value**, **tested_value**: inputs to calculate the maintainability.

3. TEST VALUES FOR EACH ONE OF THE VARIABLES PREVIOUSLY IDENTIFIED

3.1. Equivalence Classes:

— **Functional_input** = {(-∞, 0), x, x}, {[0, 10), x, x}, {[10, 35), x, x}, {[35, 50), x, x}, {[50, 70), x, x}, {[70, 90), x, x}, {[90, 100], x, x} and {(100, ∞), x, x}.

— **Maintainability_input** = {(-∞, 0), x, x, x, x}, {[0, 10), x, x, x, x}, {[10, 35), x, x, x, x}, {[35, 50), x, x, x, x}, {[50, 70), x, x, x, x}, {[70, 90), x, x, x, x}, {[90, 100], x, x, x, x} and {(100, ∞), x, x , x, x}.

— **Completeness_value** = (-∞, 0), [0, 5) and [5, ∞).
— **Correctness_value** = (-∞, 0), [0, 4), [4, 5), [5, 6) and [6, ∞).

3.2. Value Limits

— **Appropriateness_value** = 0, 2, 3, 4 and 5.
— **Min_functional** = 0, 1, 2, 3, 4 and 5.
— **Modularity_value** = 0, 1, 2, 3 and 4.
— **Reusability_value** = 0, 1, 2, 3 and 5.

3.3. Guessing Mistakes

— ***Analyzability_value*** = 0, 1, 2, 3 and 5.
— ***Modified_value*** = 0, 1, 2, 3, 4 and 5.
— ***Tested_value*** = 0, 1, 2 and 4.
— ***Min_maintainability*** = 0, 1, 2, 3, 4 and 5.

4. CALCULATE MAXIMUM POSSIBLE NUMBER OF TEST

Infinite, as some values have an infinite range of values, all possible combination of test cases reach infinity.

5. DEFINE SOME TEST SUITES USING EACH USE

Functional_input = {(-∞, 0), x, x}, {[0, 10), x, x}, {[10, 35), x, x}, {[35, 50), x, x}, {[50, 70), x, x}, {[70, 90), x, x}, {[90, 100], x, x} and {(100, ∞), x, x}.

Maintainability_input = {(-∞, 0), x, x, x, x}, {[0, 10), x, x, x, x}, {[10, 35), x, x, x, x}, {[35, 50), x, x, x, x}, {[50, 70), x, x, x, x}, {[70, 90), x, x, x, x}, {[90, 100], x, x, x, x} and {(100, ∞), x, x, x, x}.

Test suite 1 = {({0,0,0}, {0,0,0,0,0}), ({100,100,100}, {100,100,100,100,100}), ({10,35,50}, {10,35,50,70,90})}

Test suite 2 = {({100,101,0}, {-42,20,53,32,23})}

6. DEFINE TEST SUITES TO ACHIEVE PAIRWISE COVERAGE

Using last exercise test suites:

(FUNCTIONAL_INPUT, MAINTAINABILITY_INPUT)	
Pair	Visit
{0,0,0}, {0,0,0,0,0}	1
{0,0,0}, {100,100,100,100,100}	1
{0,0,0}, {10,35,50,70,90}	1
{0,0,0}, {-42,20,53,32,23}	1
{100,100,100}, {0,0,0,0,0}	1
{100,100,100}, {100,100,100,100,100}	1
{100,100,100}, {10,35,50,70,90}	1
{100,100,100}, {-42,20,53,32,23}	1
{10,35,50}, {0,0,0,0,0}	1
{10,35,50}, {100,100,100,100,100}	1

{10,35,50}, {10,35,50,70,90}	1
{10,35,50}, {-42,20,53,32,23}	1
{100,101,0}, {0,0,0,0,0}	1
{100,101,0}, {100,100,100,100,100}	1
{100,101,0}, {10,35,50,70,90}	1
{100,101,0}, {-42,20,53,32,23}	1

7. SET OF TEST CASES TO ACHIEVE COVERAGE OF DECISIONS

Our most important decision is the following one. Because it raises an exception and marks the boundaries of the program.

```
if (input < 0 || input > 100) {
    throw new IOException ();
```

Where: $A = input < 0$ and $B = input > 100$

A	B	A v B
False	False	False
False	True	True
True	False	True
True	True	True

Proposed test cases: *Input = {-10000} ,{0}, {-1}, {1}, {100}, {99}, {101} and {10000}*

8. PROPOSE TEST CASE SETS TO ACHIEVE MC/DC COVERAGE

Same decision as exercise 7

A	***B***	***A v B***	***DOMINANT CONDITION***
False	False	False	A, B
False	True	True	B
True	False	True	A
True	True	True	A, B

Proposed test cases: Input = {0}, {1}, {99}

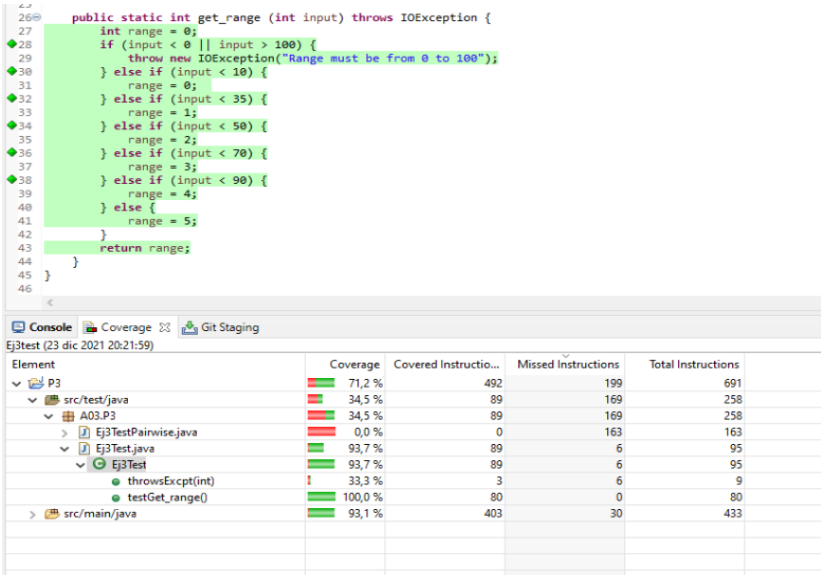
9. COMMENTS ON RESULTS

9.1. Comments For Exercise 4

For exercise 4 its coverage is 100% as all possible cases are covered

9.2. Comments For Exercise 5

Using the test cases to check the coverage we find out that a 71,5% of the program is covered



9.3. Comments For Exercise 6

Using the pairwise test class we find out that our coverage is 80,6%