**THEORICAL WORK 2 SOFTWARE ENGENEERING II**

GROUP: A03 ISO2 – 2021-22

**PROBLEM 3**

*Andrés Castellanos Cantos*

*Pablo Valverde Soriano*

[**0. STATEMENT OF THE PROBLEM** 2](#_Toc91183615)

[**1. CODE OF THE PROBLEM** 3](#_Toc91183616)

[**2. VARIABLES THAT MUST BE CONSIDERED TO TEST THE METHOD** 4](#_Toc91183617)

[**3. TEST VALUES FOR EACH ONE OF THE VARIABLES PREVIOUSLY IDENTIFIED** 4](#_Toc91183618)

[**3.1. Equivalence Classes:** 4](#_Toc91183619)

[**3.2. Value Limits** 5](#_Toc91183620)

[**3.3. Guessing Mistakes** 5](#_Toc91183621)

[**4. CALCULATE MAXIMUM POSSIBLE NUMBER OF TEST** 5](#_Toc91183622)

[**5. DEFINE SOME TEST SUITES USING EACH USE** 5](#_Toc91183623)

[**6. DEFINE TEST SUITES TO ACHIEVE PAIRWISE COVERAGE** 5](#_Toc91183624)

[**7. SET OF TEST CASES TO ACHIEVE COVERAGE OF DECISIONS** 6](#_Toc91183625)

[**8. PROPOSE TEST CASE SETS TO ACHIEVE MC/DC COVERAGE** 6](#_Toc91183626)

[**9. COMMENTS ON RESULTS** 7](#_Toc91183627)

[**9.1. Comments For Exercise 4** 7](#_Toc91183628)

[**9.2. Comments For Exercise 5** 7](#_Toc91183629)

[**9.3. Comments For Exercise 6** 7](#_Toc91183630)

# **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.

Tabla

Descripción generada automáticamenteFor functional suitability, the following table has been declared:

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.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja**Maintainability = Min {Modularity, Reusability, Analyzability, Ability to be modified, Ability to be tested}**

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):

**Tabla

Descripción generada automáticamente**

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.

Where: and

|  |  |  |
| --- | --- | --- |
| 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 |

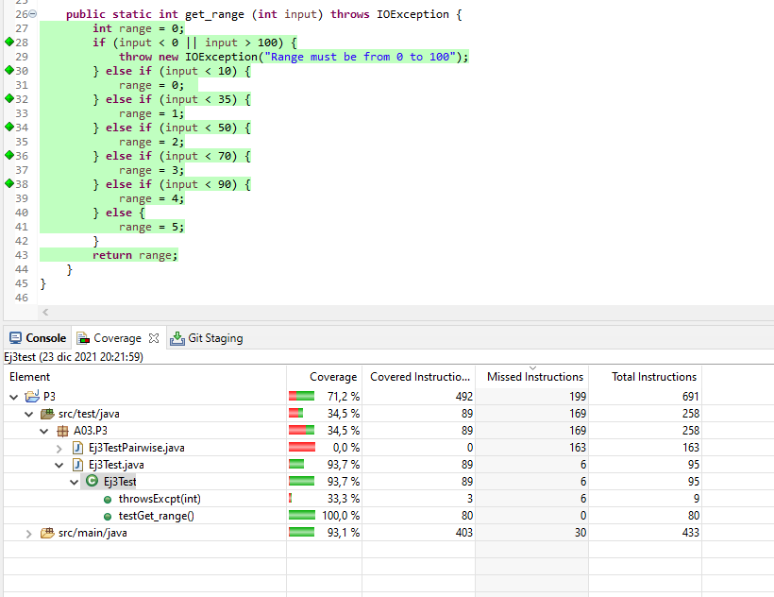
*Proposed test cases*: Input = {0}, {1}, {99}

# **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**

Texto, Tabla

Descripción generada automáticamente con confianza media Using the pairwise test class we find out that our coverage is 80,6%