Qualidade e Confiabilidade de Software
Software Testing Plan

André Macedo[1],Victor Oliveira[2]

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologias
Universidade de Coimbra
{afmacedo[1],victorho[2]}@student.dei.uc.pt

15 de Junho de 2016

Test Plan Identifier

QCS-15/16-OliveiraMacedo-1.0

Victor Oliveira
André Macedo
{victorho,afmacedo}@student.dei.uc.pt

1. Introduction

This document contains a detailed test plan, the results obtained, as well as an analysis of the aforementioned results. The test plan targets the insulin calculator software developed for the Software Quality and Dependability course, specifically the calculator developed by group constituted of João Sousa, Daniel Primo and Pedro Batista.

The software produced consists of two applications: A web service, coded in Java and a web application, coded in Python and comprised of two main modules: a client interface and a voter. The project consisted in applying the concept of N-Version programming, which is a software process methodology where different versions of a software are produced independently although utilizing the same specifications and requirements. The result is functionally equivalent software versions, but produced independently, and therefore highly likely to vary in implementation and/or other attributes.

The goal of the project was to develop an insulin calculator that utilized three distinct methods in calculating the amount of insulin doses necessary for each user. Based on each method of calculation, the application should return the amount of intake doses necessary. This was performed by utilizing a web service.

The web service that each version develops, accepts input based on each calculation method and outputs the amount of insulin intake doses.

The voter receives the results of all calculations performed (results from web services) and verifies if there exists a majority or not.

The client interface is what enables the user to communicate with the application. He may choose the type of insulin calculation, and posteriorly input the relevant data for the calculation. The three calculation types are as follows:

- Standard Mealtime Insulin Dose
    - Inputs:
        - Total grams of carbohydrates in the meal (between 60g and 120g)
        - Total grams of carbohydrates processed by 1 unit of insulin (between 10g and 15g)
        - Actual blood sugar level before meal (between 120 mg/dl and 250 mg/dl
        - Target blood sugar before meal (80 mg/dl and 120 mg/dl)
        - Individual sensitivity /between 15 mg/dl and 100 mg/dl)
    - Output:
        - Number of insulin units needed after a meal

- Background Insulin Dose
    - Input:
        - Weight in kilograms (between 40kg and 130 kg)
    - Output:
        - Number of insulin units needed between meals

- Mealtime Personal Sensitivity Insulin Dose
    - Inputs:
        - Total grams of carbohydrates in the meal (between 60g and 120g)
        - Total grams of carbohydrates processed by 1 unit of insulin (between 10g and 15g)
        - Actual blood sugar level before meal (between 120 mg/dl and 250 mg/dl
        - Target blood sugar before meal (80 mg/dl and 120 mg/dl)
        - Today's physical activity level (between 0 and 10)
        - K samples of physical activity (between 0 and 10, regarding prior days)
        - K samples of drops in blood sugar from one unit of insulin that day (between 15 mg/dl and 100 mg/dl)
    - Output:
        - Number of insulin units needed after a meal

Note: Due to the design of the system interface, the calculation of the mealtime personal sensitivity insulin dose is made by calculating the personal sensitivity first and using the output value in the mealtime standard insulin dose.

2. Software risk issues

As has been described, the developed software is of highly critical nature. This is because a wrong calculation may result in serious injuries to the user. We can then infer from these statements that the main quality attribute of the system as a whole is dependability. As a result of guaranteeing the system's dependability, N-Version programming was selected as the software development methodology. A further consequence of utilizing N-Version programming is employing a voter to verify the various sources' results, and utilizing a model checker to ensure the correctness of the voter.

Concerns regarding the voter can be directly mapped to its correctness (ensured by the model checker) and the mechanism used for determining the majority. One unit of absolute freedom was allowed in the process of determining the majority, this allowed to mitigate minor rounding errors between the different versions (i.e. the maximum difference between all of the web service results obtained, is one unit). As defined in non-functional requirement NFR1, in the event of not obtaining a majority, the system should not produce any result whatsoever.

Regarding the client interface, there is a risk as to the inputs the user submits. This risk was addressed in the requirements. According to the functional requirements defined in FR1, FR2,

FR3 and FR4, all invalid inputs must prohibit the user from being able to submit a calculation, and in the case of switching between different calculations, all prior inputs must be erased. The proper implementation of these functional requirements guarantee that invalid inputs will not be submitted as far as the user is involved. This, in turn, is directly related to the mitigation of this risk. Under most circumstances, it can be assumed that this aspect poses the lesser of risks of the system.

As to the web services, there is a concern regarding synchronization issues. As multiple web services are being employed to calculate a result, the timely response of each one is a risk regarding to having a correctly functioning system. To address this issue, non-functional requirements NFR2 and NFR3 give the system a four second timeframe to establish a connection with the web service, retrieve a result, or in the case of invalid results, give the system the rest of the time window available for another attempt (retry). This in part addresses synchronization issues as the four-second period restricts with a quantifiable number the amount of time that is valid for this operation.

3. Items and features to be tested

This test plan aims to test the modules of the insulin calculator. This consists in the testing of the web service, the voter, and the client interface. As to the fundamental tests:

- Sistem
    - Concerning the system, inputs will be verified to ensure the correct identification and handling of incorrect inputs. Incorrect inputs consist in invalid input types, or valid input types but out of the defined ranges. Also, the correctness of the results will be checked. This is performed by comparing the result values against the expected results.

- Voter
    - Particular attention to the voter must be given as to the output results. A wide range of inputs must be tested and the according outputs verified. The reasons for the importance of this test has been explained in section 2, and lie in the inherent harm that can be inflicted on erroneous results.

- Webservice
    - The web service poses a risk to the correct functioning of the application. It must be added that although invalid results in this module affect the functioning of the system, it is ultimately the voter that must produce a correct result. When testing the web service, each of the three calculation methods must be separately tested.

4. Items and features not to be tested

Here are described the tests that this test plan will *not* do. The voter will be tested only for majority/non majority functionality. In addition, extra implementation details will be left out. The test should stress the importance of how and if the voter achieves and determines if there exists a majority or not. This means all communication and other details will be left out.

For this test plan, interface testing was left out. This based on the fact that the system as a whole will be tested, which in turn rely on the interface itself to perform the actual test. Even though system testing does not substitute interface testing, it is possible to cover a considerable amount of interface functionality in this way.
Another aspect that will be left out is the production environment. Tests will be performed to the modules, but testing on multiple machine types and architectures in infeasible.

5. Testing Approach

The testing approach will base itself on black box and white box testing. In order to conduct unit tests, JUnit will be used for web service testing purposes, as the code is in Java. To test the voter, the unittest module for python will be used. In addition to these, system testing will be done manually as to simulate the end user. As mentioned earlier, this will allow us to both test the system, and a sizeable part of the interface also.

5.1. Black Box

The first step in the black box testing process is to design equivalence classes. These classes are used to divide the inputs into two groups, the valid inputs, and the invalid inputs. These two groups are then further divided into subgroups in an attempt to ignore inputs that can be considered valid. This division in the majority of the situations is performed by low value equivalence classes and high value equivalence classes. After this division is performed, boundary values are selected. These values are used as it is precisely near these values that most errors are likely to occur. To reduce the test space, random values are used instead of all possible combinations possible, as this last possibility would be infeasible to execute.

5.1.1. System

Here our main priority is to test whether the user can submit a query, in the case of the existence of out of range values. This testing is done manually through the interface. In the case of valid inputs, the result is also tested against the expected result.

5.1.2. Voter

To test the voter, the section of source code responsible for the actual voting process was tested in an isolated environment in order to confirm the correctness of the results. The test was performed using simulated inputs, and comparing the voter results with the expected results which were calculated by hand.

5.1.3. Webservice

Black box testing performed on the web service aims to verify that it doesn't accept invalid inputs, and verify that the inputs given and the outputs received are in consensus.

5.2. White Box

In order to test the voter and check for the independent paths, the control flux diagram was made. From this diagram we can identify the critical areas of the code and prepare accordingly. This allows us to see what tests are needed in order to achieve maximum coverage.

5.2.1. Voter

Here we pretend to verify whether the voters behaviour remains as predicted when invoking it with different values. The different values employed were carefully chosen as to ensure the coverage of all of the code.

5.2.2. Webservice

Here we pretend to verify whether the behaviour of the web service remains as expected when submitting different values. The test cases employed ensured that all parts of the code was passed at least once.

6. Item Pass / Fail Criteria

For the test phase, we consider any major defect to be a defect in the software that ultimately contributes to a wrong result. This is based on the fact that any erroneous calculation is potentially life threatening to the user. Therefore any defect which proves to contribute to a faulty result is considered a major defect.

Any errors in the software that hinder the correct functioning of the system, that affect any system requirements, or produce an error of any kind, but that do not contribute to errors in calculation, will be considered minor defects.

In the event of the detection of any major defect, this is sufficient to fail the test. In effect, the absence of major defects is necessary to ensure that the test passes. As for minor defects, a maximum of 4 defects is allowed, any higher value will be considered a failure.

7. Test Deliverables

The test deliverables consist of various entities. The actual test plan itself is the most obvious. This contains the actual test plan, and is a must read for anyone who will develop the tests and execute the tests. Next, all the unit tests performed on the modules will also be included. This includes the code utilized to perform the unit tests, with corresponding description and target module. All of the test cases must all be included, along with a description as what part of the system is being tested. The coverage tests pertaining to the web services are included, although annexed in the form of a web page. This contains detailed information on branch and coverability ratios for each part of the code.

8. Environmental Needs

In order to perform the aforementioned tests there are some needs inherent to the testing process. The following tools were used:

- Netbeans 8
- Sublime text 3
- JUnit
- TikiOne JaCoCoverage netbeans plugin
- Unittest python module

9. Staffing and responsibilities

In order to perform the present test plan, a group of two people is necessary to carry out the task. One role is that of the Test Lead. His responsibilities include all of the planning involved with creating and executing a test plan, including monitoring its execution.

The other role is that of a Test Engineer. His responsibilities include reading all documentation pertaining to the application, and thoroughly understand what is to be tested. Develop test cases, execute them and report on defects encountered.

In our scenario, both of the authors will perform both roles. As Test Leads, the authors will both participate in the planning of all tests. As Test Engineers, both will contribute to test development and will divide test executions between the two.

In this way, both can contribute with their insights to ensure the proper planning, execution, and reporting of the test plan.

10. Test Completion Report

10.1. Black Box

10.1.1. System

For the system tests, 7 major defects were encountered. All of the defects are from the calculus of the mealtime personal sensitivity insulin dose. In the tests, and since the standard calculation was already covered by the first set of tests, only the personal sensitivity value was considered.
19 tests were performed in the calculation of the mealtime standard insulin dose and 7 tests were performed in the calculation of the background insulin dose. Both sets of tests passed without any major or minor defect.

10.1.2. Voter

We considered the 3 errors encountered in the voter tests to be minor defects, due to the fact that given the inputs, the output is reasonably acceptable. This means that although the results are wrong, in practice it doesn't affect the user when it comes to his health. Only an invalid array size and an array of floats were given a result. The input with invalid characters does not provide an output.

10.1.3. Webservice

1 minor defect and 6 major defects were found in the webservice tests. All tests passed except for the personal sensitivity calculation, where 1 output was within a range of 1 unit (minor defect) and 6 of them were completely out of range.

10.2. White Box

10.2.1. Webservice

Although the actual value of coverage is 87%, we consider the real value to be 100%. The reason for this is the existence of code related to catching exceptions. These areas of code are never accessed in practice, and contribute to the overall decrease in the coverability ratio.

10.2.2. Voter

The critical sections of code were labelled according to the paths possible to traverse. Therefore, contiguous sections of code are labelled as a single entity, and sections where jumps in code are possible are labelled independently.

10.3. Conclusions

As shown above, the calculation of the personal sensitivity by both the system and the webservice present risks to the user. In this way, we can conclude that the system is not ready to be deployed in production. The system and the webservice should be submitted a new phase were the defects encountered can be fixed (bug fixing) and posteriorly retested.

11. Attachments
11.1. Black Box
11.1.1. System
11.1.1.1. Mealtime Standard insulin dose
11.1.1.1.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VSEC1	[60, 90[	ISEC1	]-∞, 60[
		VSEC2	[90, 120[	ISEC2	]120, +∞[
				ISEC3	Non-integer Values
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VSEC3	[10, 12[	ISEC4	]-∞, 10[
		VSEC4	12	ISEC5	]15, +∞[
		VSEC5	]12, 15]	ISEC6	Non-integer Values
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VSEC6	[120, 185[	ISEC7	]-∞, 120[
		VSEC7	[185, 250]	ISEC8	]250, +∞[
				ISEC9	Non-integer Values
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VSEC8	[80, 100[	ISEC10	]-∞, 80[
		VSEC9	[100, 120[	ISEC11	]120, +∞[
				ISEC12	Non-integer Values
P5 = Individual Sensitivity (mg/dl)	An integer P5 such that: 15 <= P5 <= 100	VSEC10	[15, 50[	ISEC13	]-∞, 15[
		VSEC11	50	ISEC14	]100, +∞[
		VSEC12	]50, 100]	ISEC15	Non-integer Values

11.1.1.1.2. Boundary values

Description	Input	Valid E.C.	Valid Boudary Values	Invalid E.C.	Invalid Boundary Values
		ID		ID	
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VSEC1	60, 90, 120	ISEC1	59
		VSEC2		ISEC2	121
				ISEC3	1.1, 'test'
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VSEC3	10, 11	ISEC4	9
		VSEC4	12	ISEC5	16
		VSEC5	13, 15	ISEC6	1.1, 'test'
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VSEC6	120, 185, 250	ISEC7	119
		VSEC7		ISEC8	251
				ISEC9	1.1, 'test'
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VSEC8	80, 100, 120	ISEC10	79
		VSEC9		ISEC11	121
				ISEC12	1.1, 'test'
P5 = Individual Sensitivity (mg/dl)	An integer P5 such that: 15 <= P5 <= 100	VSEC10	15, 49	ISEC13	14
		VSEC11	50	ISEC14	101
		VSEC12	51, 100	ISEC15	1.1, 'test'

11.1.1.1.3. Test cases

ID	Input					Observations	Expected Result	Result	Pass	Fail
	P1	P2	P3	P4	P5					
TS1	60	10	120	80	15	Valid lower boundaries	23	23	X	
TS2	120	15	250	120	100	Valid upper boundaries	5	5	X	
TS3	60	12	200	100	25	Valid random values	14	14	X	
TS4	120	14	170	100	60	Valid random values	8	8	X	
TS5	59	12	180	100	50	Invalid P1 lower boundary	IDNA	IDNA	X	
TS6	90	9	180	100	50	Invalid P2 lower boundary	IDNA	IDNA	X	
TS7	90	12	119	100	50	Invalid P3 lower boundary	IDNA	IDNA	X	
TS8	90	12	180	79	50	Invalid P4 lower boundary	IDNA	IDNA	X	
TS9	90	12	180	100	14	Invalid P5 lower boundary	IDNA	IDNA	X	
TS10	121	12	180	100	50	Invalid P1 upper boundary	IDNA	IDNA	X	
TS11	90	16	180	100	50	Invalid P2 upper boundary	IDNA	IDNA	X	
TS12	90	12	251	100	50	Invalid P3 upper boundary	IDNA	IDNA	X	
TS13	90	12	180	121	50	Invalid P4 upper boundary	IDNA	IDNA	X	
TS14	90	12	180	100	101	Invalid P5 upper boundary	IDNA	IDNA	X	
TS15	't'	12	180	100	50	Invalid input	IDNA	IDNA	X	
TS16	90	""	180	100	50	Invalid input	IDNA	IDNA	X	
TS17	90	12	180-	100	50	Invalid input	IDNA	IDNA	X	
TS18	90	12	180	80.1	50	Invalid input	IDNA	IDNA	X	
TS19	90	12	180	100	'nt'	Invalid input	IDNA	IDNA	X	

IDNA – interface does not allow

11.1.1.2. Background insulin dose

11.1.1.2.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P6 = Weight in kilograms	An integer P6 such that: 40 <= P6 <= 130	VBEC1	[40, 60]	IBEC1	]-∞, 40[
		VBEC2	]60, 90]	IBEC2	]130, +∞[
		VBEC3	]90, 130]	IBEC3	Non-integer Values

11.1.1.2.2. Boundary values

Description	Input	Valid E.C.	Valid Boudary Values	Invalid E.C.	Invalid Boundary Values
		ID		ID	
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VBEC1	40, 60	IBEC1	39
		VBEC2	61, 90	IBEC2	131
		VBEC3	91, 130	IBEC3	1.1, 'test'

11.1.1.2.3. Test cases

ID	Input	Observations	Expected Result	Result	Pass	Fail
	P6					
TB1	40	Valid lower boundariy	11	11	X	
TB2	130	Valid upper boundariy	36	36	X	
TB3	60	Valid medium values	17	17	X	
TB4	90	Valid medium values	25	25	X	
TB5	39	Invalid lower boundary	IDNA	IDNA	X	
TB6	131	Invalid upper boundary	IDNA	IDNA	X	
TB7	'test'	Invalid input	IDNA	IDNA	X	

IDNA – interface does not allow

11.1.1.3. Mealtime Personal insulin dose

11.1.1.3.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VPEC1	[60, 90[	IPEC1	]-∞, 60[
		VPEC2	[90, 120[	IPEC2	]120, +∞[
				IPEC3	Non-integer Values
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VPEC3	[10, 12[	IPEC4	]-∞, 10[
		VPEC4	12	IPEC5	]15, +∞[
		VPEC5	]12, 15]	IPEC6	Non-integer Values
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VPEC6	[120, 185[	IPEC7	]-∞, 120[
		VPEC7	[185, 250]	IPEC8	]250, +∞[
				IPEC9	Non-integer Values
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VPEC8	[80, 100[	IPEC10	]-∞, 80[
		VPEC9	[100, 120[	IPEC11	]120, +∞[
				IPEC12	Non-integer Values
P7 = Today's physical activity level	An integer P7 such that: 0 <= P7 <= 10	VPEC10	[0, 5]	IPEC13	]-∞, 0[
		VPEC11	]5, 10]	IPEC14	]10, +∞[
				IPEC15	Non-integer Values
P8 = Samples of physical activity level	An integer P8 such that: 0 <= P8 <= 10	VPEC12	[0, 5]	IPEC16	]-∞, 0[
		VPEC13	]5, 10]	IPEC17	]10, +∞[
				IPEC18	Non-integer Values
P9 = Samples of drops in blood sugar from one unit of insulin	An integer P9 such that: 15 <= P9 <= 100	VPEC14	[15, 60]	IPEC19	]-∞, 15[
		VPEC15	]60, 100]	IPEC20	]100, +∞[
				IPEC21	Non-integer Values
Size of P8	An integer such that: 2 <= P8 <= 10	VPEC16	[2, 6]	IPEC22	]-∞, 2[
		VPEC17	]6, 10]	IPEC23	]10, +∞[
				IPEC24	Non-integer Values
Size of P9	An integer such that: 2 <= P9 <= 10	VPEC18	[2, 6]	IPEC25	]-∞, 2[
		VPEC19	]6, 10]	IPEC26	]10, +∞[
				IPEC27	Non-integer Values
Samples size	An integer between 2 and 10	VPEC20	Size of P8 = size of P9	IPEC28	Size of P8 != size of P9

11.1.1.3.2. Boundary values

Description	Input	Valid E.C. ID	Valid Boudary Values	Invalid E.C. ID	Invalid Boundary Values
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VPEC1	60, 90, 120	IPEC1	59
		VPEC2		IPEC2	121
				IPEC3	1.1, 'test'
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VPEC3	10, 11	IPEC4	9
		VPEC4	12	IPEC5	16
		VPEC5	13, 15	IPEC6	1.1, 'test'
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VPEC6	120, 185, 250	IPEC7	119
		VPEC7		IPEC8	251
				IPEC9	1.1, 'test'
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VPEC8	80, 100, 120	IPEC10	79
		VPEC9		IPEC11	121
				IPEC12	1.1, 'test'
P7 = Today's physical activity level	An integer P7 such that: 0 <= P7 <= 10	VPEC10	0, 5, 10	IPEC13	-1
		VPEC11		IPEC14	11
				IPEC15	1.1, 'test'
P8 = Samples of physical activity level	An integer P8 such that: 0 <= P8 <= 10	VPEC12	0, 5,10	IPEC16	-1
		VPEC13		IPEC17	11
				IPEC18	1.1, 'test'
P9 = Samples of drops in blood sugar from one unit of insulin	An integer P9 such that: 15 <= P9 <= 100	VPEC14	15, 60, 100	IPEC19	14
		VPEC15		IPEC20	101
				IPEC21	1.1, 'test'
Size of P8	An integer such that: 2 <= P8 <= 10	VPEC16	2, 6, 10	IPEC22	1
		VPEC17		IPEC23	11
				IPEC24	1.1, 'test'
Size of P9	An integer such that: 2 <= P9 <= 10	VPEC18	2, 6, 10	IPEC25	1
		VPEC19		IPEC26	11
				IPEC27	1.1, 'test'
Samples size	An integer between 2 and 10	VPEC20	2, 10	IPEC28	1, 11

11.1.1.3.3. Test cases

ID	Input			Observations	Expected Result	Result	Pass	Fail
	P7	P8	P9					
TP1	0	[0, 0, 0, 0, 0]	[15, 15, 15, 15, 15]	Valid lower boundaries	0	0	X	
TP2	10	[10, 10, 10, 10, 10]	[100, 100, 100, 100, 100]	Valid upper boundaries	0	0	X	
TP3	5	[1, 3, 5, 7, 10]	[15, 35, 55, 75, 100]	Valid ascending values	54	54	X	
TP4	5	[7, 6, 5, 4, 3]	[80, 70, 60, 50, 40]	Valid descending values	60	60	X	
TP5	6	[2, 8]	[32, 83]	Valid random values	66	66	X	
TP6	4	[1, 6, 8, 9]	[32, 61, 91, 88]	Valid random values	53	53	X	
TP7	5	[5]	[60]	Invalid size of P8 and P9	IDNA	IDNA		
TP8	5	[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]	[60, 60, 60, 60, 60, 60, 60, 60, 60, 60]	Invalid size of P8 and P9	IDNA	0		X
TP9	-1	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid P7 lower boundary	IDNA	IDNA		
TP10	5	[-1, -1, -1, -1, -1]	[60, 60, 60, 60, 60]	Invalid P8 lower boundary	IDNA	-1		X
TP11	5	[5, 5, 5, 5, 5]	[14, 14, 14, 14, 14]	Invalid P9 lower boundary	IDNA	0		X
TP12	11	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid P7 upper boundary	IDNA	0		X
TP13	5	[11, 11, 11, 11, 11]	[60, 60, 60, 60, 60]	Invalid P8 upper boundary	IDNA	-1		X
TP14	5	[5, 5, 5, 5, 5]	[101, 101, 101, 101, 101]	Invalid P9 upper boundary	IDNA	0		X
TP15	5	[5, 5, 5, 5]	[60, 60, 60, 60, 60, 60]	Invalid P8 size != P9 size	IDNA	IDNA	X	
TP16	't'	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid input	IDNA	IDNA	X	
TP17	5	["t", "e", "s", "t", "s"]	[60, 60, 60, 60, 60]	Invalid input	IDNA	-1		X

IDNA – interface does not allow

11.1.2. Voter

11.1.2.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
Majority is reached	An array with 3 results	VVEC1	]-1, +∞[	IVEC1	]-∞, -1]
				IVEC2	Non-integer Values
Size of results array	An array with 3 or more results	VVEC2	[3, +∞[	IVEC3	]-∞, 3[

11.1.2.2. Boundary values

Description	Input	Valid E.C.	Valid Boudary Values	Invalid E.C.	Invalid Boundary Values
		ID		ID	
Majority is reached	An array with 3 results	VVEC1	]-1, +∞[	IVEC1	-1
				IVEC2	1.1, 'test'
Size of results array	An array with 3 or more results	VVEC2	3, 5	IVEC3	1, 2

11.1.2.3. Test cases

ID	Input	Observations	Expected Result	Result	Pass	Fail
TV1	[2, 2, 2]	Valid majority result	2	2	X	
TV2	[1, 2, 3]	Valid majority result	2	2	X	
TV3	[-1, 3, 3]	Valid majority result	3	3	X	
TV4	[2, 2, 5, 5, 5]	Valid majority result	5	5	X	
TV5	[-1, 2, 4]	Invalid majority result	-1	-1	X	
TV6	[1, 3, 5]	Invalid majority result	-1	-1	X	
TV7	[2, 2]	Invalid array size	-1	2		X
TV8	[2]	Invalid array size	-1	-1	X	
TV9	[7.5, 7.5, 7.5]	Invalid input	-1	7.5		X
TV10	['t', 'e', 's']	Invalid input	-1	Error		X

11.1.3. Webservice

11.1.3.1. Mealtime standard insulin dose

11.1.3.1.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VSEC1	[60, 90[	ISEC1	]-∞, 60[
		VSEC2	[90, 120[	ISEC2	]120, +∞[
				ISEC3	Non-integer Values
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VSEC3	[10, 12[	ISEC4	]-∞, 10[
		VSEC4	12	ISEC5	]15, +∞[
		VSEC5	]12, 15]	ISEC6	Non-integer Values
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VSEC6	[120, 185[	ISEC7	]-∞, 120[
		VSEC7	[185, 250]	ISEC8	]250, +∞[
				ISEC9	Non-integer Values
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VSEC8	[80, 100[	ISEC10	]-∞, 80[
		VSEC9	[100, 120[	ISEC11	]120, +∞[
				ISEC12	Non-integer Values
P5 = Individual Sensitivity (mg/dl)	An integer P5 such that: 15 <= P5 <= 100	VSEC10	[15, 50[	ISEC13	]-∞, 15[
		VSEC11	50	ISEC14	]100, +∞[
		VSEC12	]50, 100]	ISEC15	Non-integer Values

11.1.3.1.2. Boundary values

Description	Input	Valid E.C. ID	Valid Boudary Values	Invalid E.C. ID	Invalid Boundary Values
P1 = Total grams of carbohydrates in the meal (g)	An integer M1 such that: 60 <= P1 <= 120	VSEC1	60, 90, 120	ISEC1	59
				ISEC2	121
		VSEC2		ISEC3	1.1, 'test'
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VSEC3	10, 11	ISEC4	9
		VSEC4	12	ISEC5	16
		VSEC5	13, 15	ISEC6	1.1, 'test'
P3 = Actual blood sugar level measured before the meal (mg/dl)	An integer P3 such that: 120 <= P3 <= 250	VSEC6	120, 185, 250	ISEC7	119
				ISEC8	251
		VSEC7		ISEC9	1.1, 'test'
P4 = Target blood sugar before the meal (mg/dl)	An integer P4 such that: 80 <= P4 <= 120	VSEC8	80, 100, 120	ISEC10	79
				ISEC11	121
		VSEC9		ISEC12	1.1, 'test'
P5 = Individual Sensitivity (mg/dl)	An integer P5 such that: 15 <= P5 <= 100	VSEC10	15, 49	ISEC13	14
		VSEC11	50	ISEC14	101
		VSEC12	51, 100	ISEC15	1.1, 'test'

11.1.3.1.3. Test cases

ID	Input					Observations	Expected Result	Result	Pass	Fail
	P1	P2	P3	P4	P5					
TS1	60	10	120	80	15	Valid lower boundaries	23	23	X	
TS2	120	15	250	120	100	Valid upper boundaries	5	4	X	
TS3	60	12	200	100	25	Valid random values	14	14	X	
TS4	120	14	170	100	60	Valid random values	8	8	X	
TS5	59	12	180	100	50	Invalid P1 lower boundary	-1	-1	X	
TS6	90	9	180	100	50	Invalid P2 lower boundary	-1	-1	X	
TS7	90	12	119	100	50	Invalid P3 lower boundary	-1	-1	X	
TS8	90	12	180	79	50	Invalid P4 lower boundary	-1	-1	X	
TS9	90	12	180	100	14	Invalid P5 lower boundary	-1	-1	X	
TS10	121	12	180	100	50	Invalid P1 upper boundary	-1	-1	X	
TS11	90	16	180	100	50	Invalid P2 upper boundary	-1	-1	X	
TS12	90	12	251	100	50	Invalid P3 upper boundary	-1	-1	X	
TS13	90	12	180	121	50	Invalid P4 upper boundary	-1	-1	X	
TS14	90	12	180	100	101	Invalid P5 upper boundary	-1	-1	X	
TS15	't'	12	180	100	50	Invalid input	Error	Error	X	
TS16	90	""	180	100	50	Invalid input	Error	Error	X	
TS17	90	12	180-	100	50	Invalid input	Error	Error	X	
TS18	90	12	180	80.1	50	Invalid input	Error	Error	X	
TS19	90	12	180	100	'nt'	Invalid input	Error	Error	X	

11.1.3.2. Background insulin dose

11.1.3.2.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P6 = Weight in kilograms	An integer P6 such that: 40 <= P6 <= 130	VBEC1	[40, 60]	IBEC1	]-∞, 40[
		VBEC2	]60, 90]	IBEC2	]130, +∞[
		VBEC3	]90, 130]	IBEC3	Non-integer Values

11.1.3.2.2. Boundary values

Description	Input	Valid E.C.	Valid Boudary Values	Invalid E.C.	Invalid Boundary Values
		ID		ID	
P2 = Total grams of carbohydrates processed by 1 unit of rapid acting insulin (g)	An integer P2 such that: 10 <= P2 <= 15	VBEC1	40, 60	IBEC1	39
		VBEC2	61, 90	IBEC2	131
		VBEC3	91, 130	IBEC3	1.1, 'test'

11.1.3.2.3. Test cases

ID	Input	Observations	Expected Result	Result	Pass	Fail
	P6					
TB1	40	Valid lower boundariy	11	11	X	
TB2	130	Valid upper boundariy	36	36	X	
TB3	60	Valid medium values	17	17	X	
TB4	90	Valid medium values	25	25	X	
TB5	39	Invalid lower boundary	-1	-1	X	
TB6	131	Invalid upper boundary	-1	-1	X	
TB7	50.5	Invalid input	Error	Error	X	

11.1.3.3. Personal Sesitivity

11.1.3.3.1. Equivalence classes

Description	Input	Valid E.C.		Invalid E.C.	
		ID	Class Definition	ID	Class Definition
P7 = Today's physical activity level	An integer P7 such that: 0 <= P7 <= 10	VPEC10	[0, 5]	IPEC13	]-∞, 0[
		VPEC11	]5, 10]	IPEC14	]10, +∞[
				IPEC15	Non-integer Values
P8 = Samples of physical activity level	An integer P8 such that: 0 <= P8 <= 10	VPEC12	[0, 5]	IPEC16	]-∞, 0[
		VPEC13	]5, 10]	IPEC17	]10, +∞[
				IPEC18	Non-integer Values
P9 = Samples of drops in blood sugar from one unit of insulin	An integer P9 such that: 15 <= P9 <= 100	VPEC14	[15, 60]	IPEC19	]-∞, 15[
		VPEC15	]60, 100]	IPEC20	]100, +∞[
				IPEC21	Non-integer Values
Size of P8	An integer such that: 2 <= P8 <= 10	VPEC16	[2, 6]	IPEC22	]-∞, 2[
		VPEC17	]6, 10]	IPEC23	]10, +∞[
				IPEC24	Non-integer Values
Size of P9	An integer such that: 2 <= P9 <= 10	VPEC18	[2, 6]	IPEC25	]-∞, 2[
		VPEC19	]6, 10]	IPEC26	]10, +∞[
				IPEC27	Non-integer Values
Samples size	An integer between 2 and 10	VPEC20	Size of P8 = size of P9	IPEC28	Size of P8 != size of P9

11.1.3.3.2. Boundary values

Description	Input	Valid E.C. ID	Valid Boudary Values	Invalid E.C. ID	Invalid Boundary Values
P7 = Today's physical activity level	An integer P7 such that: 0 <= P7 <= 10	VPEC10	0, 5, 10	IPEC13	-1
		VPEC11		IPEC14	11
				IPEC15	1.1, 'test'
P8 = Samples of physical activity level	An integer P8 such that: 0 <= P8 <= 10	VPEC12	0, 5,10	IPEC16	-1
		VPEC13		IPEC17	11
				IPEC18	1.1, 'test'
P9 = Samples of drops in blood sugar from one unit of insulin	An integer P9 such that: 15 <= P9 <= 100	VPEC14	15, 60, 100	IPEC19	14
		VPEC15		IPEC20	101
				IPEC21	1.1, 'test'
Size of P8	An integer such that: 2 <= P8 <= 10	VPEC16	2, 6, 10	IPEC22	1
		VPEC17		IPEC23	11
				IPEC24	1.1, 'test'
Size of P9	An integer such that: 2 <= P9 <= 10	VPEC18	2, 6, 10	IPEC25	1
		VPEC19		IPEC26	11
				IPEC27	1.1, 'test'
Samples size	An integer between 2 and 10	VPEC20	2, 10	IPEC28	1, 11

11.1.3.3.3. Test cases

ID	Input			Observations	Expected Result	Result	Pass	Fail
	P7	P8	P9					
TP1	0	[0, 0, 0, 0, 0]	[15, 15, 15, 15, 15]	Valid lower boundaries	0	0	X	
TP2	10	[10, 10, 10, 10, 10]	[100, 100, 100, 100, 100]	Valid upper boundaries	0	100		X
TP3	5	[1, 3, 5, 7, 10]	[15, 35, 55, 75, 100]	Valid ascending values	54	41		X
TP4	5	[7, 6, 5, 4, 3]	[80, 70, 60, 50, 40]	Valid descending values	60	59		X
TP5	6	[2, 8]	[32, 83]	Valid random values	66	30		X
TP6	4	[1, 6, 8, 9]	[32, 61, 91, 88]	Valid random values	53	28		X
TP7	5	[5]	[60]	Invalid size of P8 and P9	-1	0		X
TP8	5	[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]	[60, 60, 60, 60, 60, 60, 60, 60, 60, 60]	Invalid size of P8 and P9	-1	60		X
TP9	-1	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid P7 lower boundary	-1	-1	X	
TP10	5	[-1, -1, -1, -1, -1]	[60, 60, 60, 60, 60]	Invalid P8 lower boundary	-1	-1	X	
TP11	5	[5, 5, 5, 5, 5]	[14, 14, 14, 14, 14]	Invalid P9 lower boundary	-1	-1	X	
TP12	11	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid P7 upper boundary	-1	-1	X	
TP13	5	[11, 11, 11, 11, 11]	[60, 60, 60, 60, 60]	Invalid P8 upper boundary	-1	-1	X	
TP14	5	[5, 5, 5, 5, 5]	[101, 101, 101, 101, 101]	Invalid P9 upper boundary	-1	-1	X	
TP15	5	[5, 5, 5, 5]	[60, 60, 60, 60, 60, 60]	Invalid P8 size != P9 size	-1	-1	X	
TP16	't'	[5, 5, 5, 5, 5]	[60, 60, 60, 60, 60]	Invalid input	Error	Error	X	
TP17	5	["t", "e", "s", "t", "s"]	[60, 60, 60, 60, 60]	Invalid input	Error	Error	X	

11.2. White Box

11.2.1. Webservice

11.2.1.1. Coverage

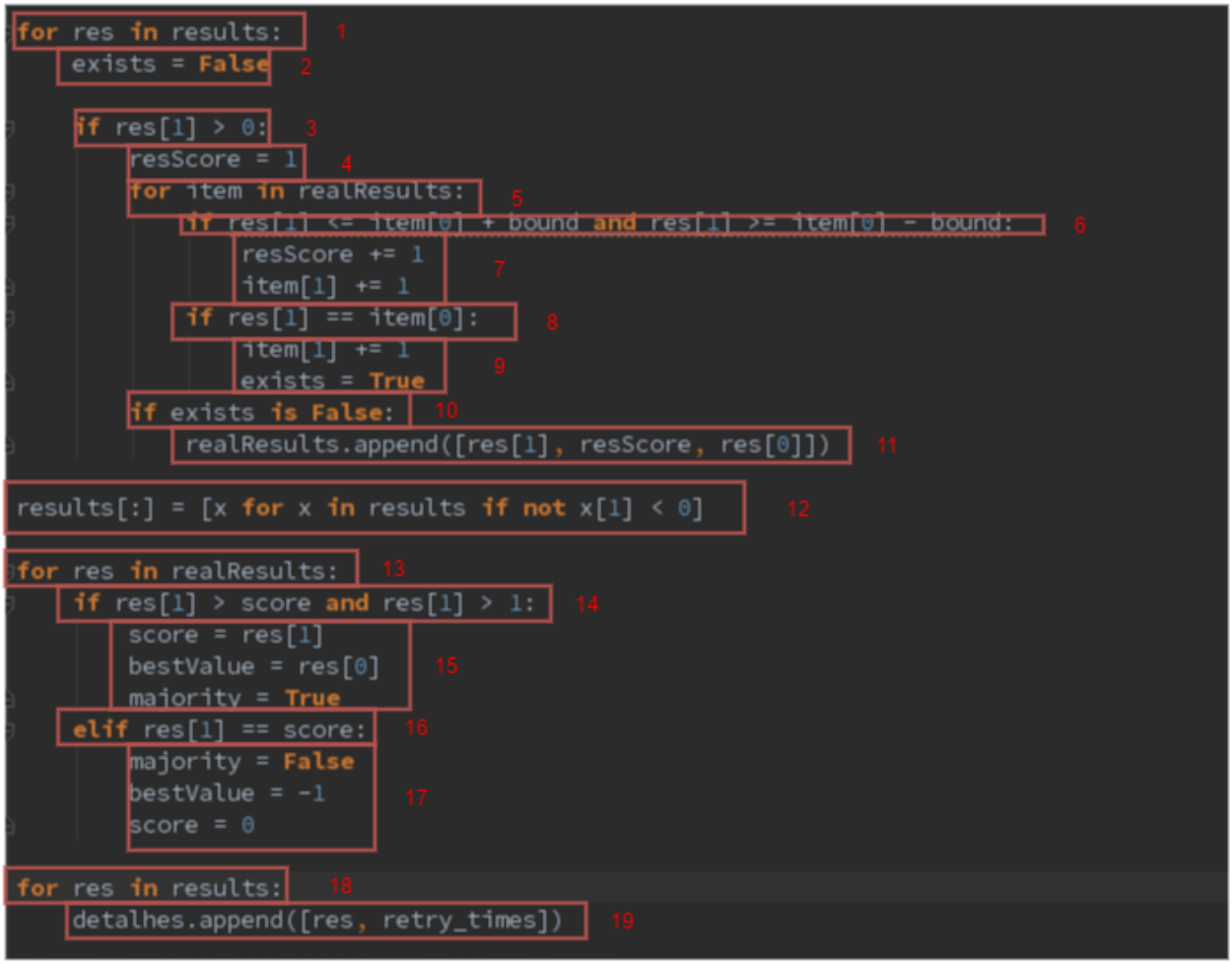
The coverage results for the webservice can checked in the folder “coverage”

11.2.2. Voter

11.2.2.1. Coverage

```
N_WebServices = 3
custom_timeout = 4
bound = 1
majority = False
wsdlFiles = [...]
runtime_wsdlList = wsdlFiles
results = [[1, 1], [7, 7], [7, 7]]
detalhes = []
time_flag = False
bestValue = -1
score = 0
retry_times = 0
realResults = []

for res in results:
    exists = False

    if res[1] > 0:
        resScore = 1
        for item in realResults:
            if res[1] <= item[0] + bound and res[1] >= item[0] - bound:
                resScore += 1
                item[1] += 1
            if res[1] == item[0]:
                item[1] += 1
                exists = True
        if exists is False:
            realResults.append([res[1], resScore, res[0]])

results[:] = [x for x in results if not x[1]<0]

for res in realResults:
    if res[1] > score and res[1] > 1:
        score = res[1]
        bestValue = res[0]
        majority = True
    elif res[1] == score:
        majority = False
        bestValue = -1
        score = 0

for res in results:
    detalhes.append([res, retry_times])

```

11.2.2.2. Critical sections of the code



11.2.2.3. Flux control diagram