Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Software Quality and Dependability

DEI-FCTUC, 2014/2015

(Updated on April 28th, as result of the reviews done by the students in the TP classes)

Assignment 1: N-Version programming web services

N-Version programming, initially proposed by Algirdas Avizienis from University of California, LA, USA, consists of the independent development of N versions (N > 2) of functionally equivalent programs (or software components) from the same specification. The goal is to compare (i.e., vote) the results produced by each program replica, in order to mask the effects of possible software faults (bugs) and assure correct output results.

N-Version programming relies on two basic assumptions. First, it considers that even using the most stringent software development methods and performing the most comprehensive tests, it is not possible to assure that the developed code is 100% free of software bugs. And second, the key to achieve reliable software is diversity, as it is unlike that independent teams (and, very often, each team uses different development methodologies, different languages, compilers, etc.) will produce code with exactly the same bugs. Thus, if more than 2 versions are executed, the majority voting of the results will assure correct results.

Although N-Version programming is used in highly critical systems (e.g., flight control system of the Airbus aircrafts), it has several problems and difficulties. The most obvious one is the very high cost, as the software development cost is multiplied by N. But the fact that the initial specification (or software requirement specification) is shared by all teams also raises concerns, as faults in the specification will affect all the replicas and cannot be tolerated. Even the fundamental N-Version assumption that software developed independently will fail independently has been challenged. In fact, there is a well-known experiment conducted by Nancy Leveson and John Knight that rejected the hypothesis of independent failures.[1]

In spite of the criticism around N-Version programming, this technique is a good learning example on how to manage redundancy to achieve high reliability. In fact, N-Version programming is the software equivalent of the classic hardware redundant architectures, such as the TMR (Triple Modular Redundancy). In TMR the result produced by three hardware modules (components or even complete systems) are voted to mask (i.e., filter out) possible incorrect outputs and assure correct results.

---

[1] See the paper here: http://sunnyday.mit.edu/papers/nver-tse.pdf

[2] Although testing will be studied in detail further on in the ***Software Quality and Dependability*** course, we trust that the students already know enough to perform an acceptable test campaign of the web service.

A well-known technical problem of voting techniques lies in the fact that the voter is a single point of failure. This is the case of N-Version programming as well. Although it is possible to design replicated voters, or even use features of the application domain to decide on the correct result (e.g., replicated displays, mechanic actuators that overpower erroneous commands, etc.), the general idea followed in N-Version programming is that the voting software should be simple enough to allow a correct implementation (e.g., by using exhaustive formal verification). In any case, even being relatively simple, it is worth saying that a typical the N-Version programming voter must take care of two paramount tasks:

- **Synchronization**:
  As each version/replica has different execution times, the voter must wait for the results of all replicas (or at least, wait for the results from 3 replicas, in cases where there are more that 3). This requires a timeout mechanism, as faults in one replica may cause a crash and in that case the replica does not produce any result. In general, the timeout is triggered by the first result that reaches the voter and the amount of time defined for the arrival of all the results is dependent on the application. Obviously, when the applications have a cyclic nature and constantly produce results, the management of all the timeouts can be rather complex.
- **Voting**:
  The majority voting is highly dependent on the output domain. When the result of each replica is deterministic the voting process is straightforward. But the process can be more complex, depending on the amount of data involved in the voting process and, in particular, when a range, instead of a single deterministic value, determines the notion of correctness of the results. In fact, in many applications the results produced by each replica may have slight variations (e.g., due to arithmetic rounding, algorithm variances, etc.), which means that the voter needs to use a band or range in the majority voting. Another difficulty in the voter implementation is related to the absence of results from one or more replicas (the timeout issue already mentioned). If the number of replicas that produce results is less than 3, then the voter can only detect disagreement when there are 2 results available (and cannot conclude anything about the result, when there is only the result from one version). In case of disagreement (i.e., no majority is reached), the voter must have a well-defined behavior. That is, the behavior of the voter (and the consequent output result) must be determined by design and must cover all the possible scenarios related to the inputs received from the replicas.

Current cloud oriented technologies and new forms of distributing services through the Internet have renewed the interest on old ideas such as N-Version programming. In fact, software applications are often built using off-the-shelf software components and software is increasingly being designed by decomposing a system into subcomponents that can be purchased from different suppliers, very often in the form of web services.

The idea of using functional equivalent web services in a voting process as proposed by the N-Version programming approach is now being considered more and more as a practical solution to increase the reliability of software results is specific critical applications.

1. Highly reliable insulin dose calculator

The purpose of the assignment is to develop a web service based N-Version programming insulin dose calculator for patients with type-2 diabetes. Patients with type-2 diabetes need periodic insulin injections to help process blood sugar. It is critical to be able to

provide enough insulin to metabolize the carbohydrates obtained from food, and even more critical to never inject higher doses that may cause hypoglycemia, which may lead to coma and brain damage.

One of the fundamental goals is to maintain a target glucose level throughout the day somewhere between 80 mg/dl and 120 mg/dl (constant for a given patient, as prescribed by the doctor). Patients regularly measure the blood glucose level and calculate the necessary insulin dose.

Patients inject regularly a certain number of ***insulin units*** throughout the day, which add up to a ***daily insulin total***. Part of the daily total insulin dose is called ***basal insulin replacement***, and is intended to cover the background need for insulin between meals. The other part is called ***bolus insulin replacement***, and is needed after each meal to process the carbohydrates obtained from food.

The goal of the insulin dose calculator to be developed in this assignment is to indicate the patient the insulin dose (i.e., number of insulin units) he/she must take at different moments of the day. The patient will input a given number of parameters (described next in the methods interface) and the system will calculate the adequate number of insulin units that the patient must take. This is, obviously, a critical application, as the incorrect insulin dose may have rather dramatic consequences for the patient.

In general terms, each group (3 students per group) should do the following to fulfill this assignment:

a. Develop one version of the web service to calculate the insulin dose and make that version available for the other student groups.

b. Develop the voting software that uses the results calculated by N versions (typically 3 versions) of functionally equivalent web services (developed by the several student groups) to produce the final insulin dose result.

c. Develop a simple web interface or a standalone application that allows the patient to input the parameters and displays the final result.

d. Write a short report describing the key aspects of the work developed.

The voter developed by each group must use the web service developed by the group to calculate the insulin dose and a number of web services developed by the other student groups (at least 3 web services in total). It is assumed that the web services reside in different machines and that the voter developed by each group of students will handle all the tasks and idiosyncrasies of N-version voting. This includes the synchronization of the web services results, the use of timeouts to resolve the cases when a given service does not respond, and the criteria to decide if the result produced by a given web service version is valid or must be excluded in the voting process. The response of the system/voter when it is not possible to reach a majority result must be planned and implemented as well.

It is worth mentioning that the use of N-Version programming to assure a reliable result through voting does not mean that the software quality of each version of the web service developed to calculate the insulin dose can be neglected. A fault tolerant technique such as N-Version programming is never a replacement for bad code or bad software engineering.

Each web service must be tested[2] carefully to assure that the code is correct, as much as possible. Additionally, the input parameters should be checked to avoid robustness failures of the web service when it is called with nonsense input parameters.

2. Specification of the insulin dose calculator

This section describes first the web service to be developed in general terms and then presents the requirement specifications of the web application (or the standalone application, if the student group prefer that) to be developed in the assignment. The requirements specification includes functional requirements, non-functional requirements and design constraints, with particular emphasis on the N-Version voter that constitutes the major design constraint.

2.1. Web service for the insulin dose calculation

The web service to be implemented by each student group has one class (InsulinDoseCalculator) and the behavior of the object is described by the following methods:

- **Web service method: mealtimeInsulinDose**

  Goal: calculates the number of insulin units needed after one meal.

  Returns: the number of units of rapid acting insulin needed after a meal (i.e., bolus insulin replacement dose).

  Inputs:

  - total grams of carbohydrates in the meal (between 60g and 120g);
  - total grams of carbohydrates processed by 1 unit of rapid acting insulin (between 10g/unit and 15g/unit, but the typical value is 12g/unit);
  - actual blood sugar level measured before the meal (between 120mg/dl and 250mg/dl);
  - target blood sugar before the meal (between 80mg/dl and 120mg/dl);
  - individual sensitivity (between 15mg/dl and 100mg/dl per unit of insulin).

  The number of units of rapid acting insulin after a meal is equal to the ***high blood sugar dose*** plus the ***carbohydrate dose***.

  The ***high blood sugar dose*** is calculated by taking the blood sugar level measured before the meal minus the target blood sugar before the meal, and dividing that amount by the amount that one unit of the used insulin will decrease in that particular individual (i.e., individual sensitivity).

  The ***carbohydrate dose*** equals the total grams of carbohydrates consumed during the meal divided by the number of grams processed by one unit of the used rapid acting insulin. This value is adjusted according to individual sensitivity, by stipulating that if one unit of insulin processes 12g of mealtime carbohydrates, it will drop blood sugar by 50 mg/dl in an average individual (proportions are maintained if the insulin used

[2] Although testing will be studied in detail further on in the ***Software Quality and Dependability*** course, we trust that the students already know enough to perform an acceptable test campaign of the web service.

processes a higher or lower value of carbohydrates, compared to the reference of 12g/unit).

- **Web service method: backgroundInsulinDose**

  Goal: calculates the total number of units of insulin needed between meals.

  Returns: Background insulin dose.

  Input: Weight in kilograms (between 40kg and 130kg).

  The general approach is to take 0.55 multiplied by the weight in kilograms as the total (i.e., mealtime plus background) daily insulin requirement. The background insulin dose is generally 50% of that total.

- **Web service method: personalSensitivityToInsulin**

  Goal: determines an individual's sensitivity to one unit of insulin.

  Returns: the drop in blood sugar resulting from one unit of insulin, in mg/dl.

  Inputs:

  - today's physical activity level (between 0 and 10);
  - K samples of physical activity level in some day (between 0 and 10);
  - K samples of drops in blood sugar from one unit of insulin in that day (between 15mg/dl and 100mg/dl).

  The method will perform a simple linear regression to predict the blood sugar drop. This method accepts two arrays of samples of physical activity level and blood sugar drop. The two arrays must therefore have the same length (i.e., number of samples) and a minimum of 2 and a maximum of 10 samples. First, a simple linear regression (least squares method) is performed to compute *alpha* and *beta*. Then, the return value is *alpha* + *beta* x physical activity level.

  The formulas to calculate the *alpha* and *beta* parameter in simple linear regression can be found in the Internet (for example, the Wikipedia has a fairly good explanation of simple linear regression: http://en.wikipedia.org/wiki/Simple_linear_regression).

2.2. Functional requirements

The web application (or the standalone application, depending on each student group choice) that will be used by the patient to obtain the insulin dose shall fulfill the following functional requirements:

- **ID: FR1**

  Title: Select type of insulin dose calculation and presents the screen to input parameters

  Description: The user shall[3] be able to select the type of insulin dose calculation required, which shall be chosen from the following list:

[3] As you know, the use of the modal verb shall in the imperative sense is quite typical in the requirement specification. It means mandatory requirements that must be implemented and tested.

- Mealtime insulin dose - standard insulin sensitivity
- Mealtime insulin dose - personal insulin sensitivity
- Background insulin dose

When a selection is made, the screen with the fields to input the appropriate parameters for each type of insulin dose calculation shall be presented. Additionally, information from previous user's actions (if any) shall be deleted.

Rational: In order for a user to select the type of insulin dose calculation.

Dependencies: None

- **ID: FR2**

  Title: Mealtime insulin dose calculation using standard insulin sensitivity

  Description: The user shall be able to input the following parameters in the appropriate fields:

  - Total grams of carbohydrates in the meal; the field is presented empty and the input value must be between 60g and 120g.
  - Total grams of carbohydrates processed by 1 unit of rapid acting insulin; the field is prefilled with 12 but the user shall be able to change the value. The input value must be between 10g/unit and 15g/unit.
  - Actual blood sugar level measured before the meal; the field is presented empty and the input value must be between 120mg/dl and 250mg/dl.
  - Target blood sugar before the meal; the field is presented empty and the input value must be between 80mg/dl and 120mg/dl.
  - Individual sensitivity; the field is prefilled with 50 but the user shall be able to change the value. The input value must be between 15mg/dl and 100mg/dl.

  The button "Calculate insulin dose" shall be inactive whenever one or more of the input fields are empty, and shall became active as soon as the user inserts valid input values in all the fields. When the user clicks on the button "Calculate insulin dose", the system shall calculate the mealtime insulin dose for standard insulin sensitivity.

  Rational: In order for a user to input the parameters and obtain the mealtime insulin dose for standard insulin sensitivity.

  Dependencies: FR1

- **ID: FR3**

  Title: Mealtime insulin dose calculation using personal insulin sensitivity

  Description: The user shall be able to input the following parameters in the appropriate fields:

  - Total grams of carbohydrates in the meal; the field is presented empty and the input value must be between 60g and 120g.
  - Total grams of carbohydrates processed by 1 unit of rapid acting insulin; the field is prefilled with 12 but the user shall be able to change the value. The input value must be between 10g/unit and 15g/unit.
  - Actual blood sugar level measured before the meal; the field is presented empty and the input value must be between 120mg/dl and 250mg/dl.
  - Target blood sugar before the meal; the field is presented empty and the input value must be between 80mg/dl and 120mg/dl.

- Today's physical activity level; the field is presented empty and the input value must be between 0 and 10. The value 0 represents no physical activity and 10 represents very high intensity of physical activity.
- Samples of physical activity level in a given day. The number of samples shall be between 2 and 10, and the values imputed for each sample must be between 0 (no physical activity) and 10 (very high physical activity).
- Samples of drops in blood sugar from one unit of insulin in that day. The number of samples shall be between 2 and 10, and the values imputed for each sample must be between 15mg/dl and 100mg/dl.

The button "Calculate insulin dose" shall be inactive whenever one or more of the input fields are empty or the number of samples of physical activity and drops in blood sugar is odd or is less than 2. The button "Calculate insulin dose" shall became active as soon as the user inserts valid input values in all the fields, including a even number of samples of physical activity and drops in blood sugar (minimum of 2 samples). When the user clicks on the button "Calculate insulin dose", the system shall calculate and display the mealtime insulin dose for personal insulin sensitivity. The individual's sensitivity to one unit of insulin is calculated using simple linear regression from the samples provided.

Rational: In order for a user to input the parameters and obtain the mealtime insulin dose for personal insulin sensitivity.

Dependencies: FR1

- **ID: FR4**

  Title: Background insulin dose calculation

  Description: The user shall be able to input the following parameter in the appropriate field:

  - Weight in kilograms; the field is presented empty and the input value must be between 40kg and 130kg.

  The button "Calculate insulin dose" shall be inactive whenever the input field is empty and shall became active as soon as a valid value is imputed. When the user clicks on the button "Calculate insulin dose", the system shall calculate and display the background insulin dose.

  Rational: In order for a user to input the parameter and obtain the background insulin dose.

  Dependencies: FR1

- **ID: FR5**

  Title: Show technical details on the insulin dose calculation

  Description: The user shall be able to see detailed information on the insulin dose calculation. After the calculation of the insulin dose (in all the types of insulin dose calculation stated in FR1), the button "Show technical information" shall became active. When the user clicks on the button "Show technical information", the system shall display the following information:

  - Number of web services (versions) used in the calculation.
  - The result (the actual insulin dose) calculated by each web service version.
  - The majority result (i.e., the final result produced by the voter) computed from the results produced by the web service versions.

If one web service version does not produce a valid result (i.e., the calculation is terminated by timeout), the technical information displayed for that web service version shall show "Timeout" instead of the numerical result (insulin dose).

Rational: This functional requirement is related to the design constraint DC1 (described further on) that states the system shall be implemented using N-Version programming. In order for a user to see detailed information about the result produced by each web service version and by the N-Version programming voter.

Dependencies: FR1, FR2, FR3, FR4

2.3. Non-functional requirements

The application that will be used by the patient to obtain the insulin dose shall fulfill the following non-functional requirements:

- **ID: NFR1**

  Title: Reliable insulin dose calculation

  Description: The insulin dose calculation result presented to the user shall be reliable, as achieved by 3 or more independent calculations and decided by a majority voter that is formally correct.

  Rational: In order for a user to receive a reliable result, or no result at all if there is no possible to establish a majority results.

  Dependencies: None

- **ID: NFR2**

  Title: Performance of insulin dose calculation

  Description: The system shall produce the insulin dose calculation result within less than 4 seconds, after the user ask the system to calculate the result from the input parameters (i.e., after the user clicks on the button "Calculate insulin dose"). If, for any reason, the system cannot calculate the result (e.g., no majority is reached within this timeframe) the interface shall display the following message: "It was not possible to calculate the insulin dose; please try again".

  Rational: In order to establish a maximum time the user shall wait for the calculation.

  Dependencies: None

- **ID: NFR3**

  Title: Retry calculations

  Description: The system shall retry a given insulin dose calculation, with the same data input by the user, if it is not possible to establish a majority voting. The retry or retries can use a different set of web service versions, if available. In case of retries, the time elapsed from the moment when the user clicks the button "Calculate insulin dose" and the final response shall be no more than 4 seconds.

  Rational: In order to improve the availability of reliable calculation within the timeframe established for the system to provide results.

  Dependencies: NFR2

2.4. Design Constraints

The design of the application to calculate the insulin dose shall fulfill the following design constraint:

- **ID: DC1**

  Title: N-Version programing

  Description: The system shall use N-Version programming.

  Dependencies: None

- **ID: DC2**

  Title: Web service technology

  Description: The system shall use web services to implement each version used in the N-Version programming voting step. Each web service available to calculate the insulin dose shall run in different machines. The web service technology to be used is the one used in the example provided as part of the specifications and assumptions of the assignment. All the web services share the same public interface (provided as the public interface of the InsulinDoseCalculator) and the different URLs of the WSDL describing the web services available are stored in the Google Drive file located at https://docs.google.com/spreadsheets/d/1nrpNjbce822EvCm8JA2MDNkFhlRxyPWCgTA8MKhoqzI/edit#gid=0. The code of the voter can use a local copy of the URLs of the WSDL of the web services used in the voting process (at least 3 web services).

  Dependencies: DC1

- **ID: DC3**

  Title: Formal verification of the voter using model checking

  Description: The voter specified and developed for the N-Version programming shall be exhaustively verified using model checking.

  Dependencies: DC1

- **ID: DC4**

  Title: Voting mechanism

  Description: The voter shall use a simple majority voting of the integer values of insulin dose returned by the web services called. Two results provided by two web services called with the same input parameters shall be considered equivalent if the difference between them is less or equal to 1 (dose of insulin). The result of the voting shall be either the majority result (i.e., the positive integer representing the correct insulin dose, including zero) or an error code (-1) if there is no majority.

  Dependencies: DC1, DC2

- **ID: DC5**

  Title: Precision of the calculations and rounding

  Description: The methods shall represent the numeric values using Java double data type and the rounding to convert the result in the integer output shall be done after all the calculations and use the standard arithmetic rounding rules.

  Dependencies: FR2, FR3 and FR4

3. Outcome

The outcome of the assignment is the **software developed** and a **written report**. As mentioned, the software developed should have the following elements:

- Web service to calculate the insulin dose. This web service must be publically available to be used by the other student groups.
- Voting code that uses the results calculated by N versions of functionally equivalent web services (developed by several student groups) to produce the final insulin dose result.
- Simple web interface or stand alone application that allows the patient to input the parameters and displays the final result.

The written report (PDF file) should describe the solution in general terms and avoid details that can be better understood by looking at the code. The report should include the following elements:

- Software architecture overview (main modules, web services, etc.).
- Description of the main modules developed by each group of students (web service developed by the group to calculate insulin dose, N-version voter, interface application). In particular, the implementation of the voter should be described with care.
- Formal verification of the voter.

There is no suggested template for the written report structure. The goal is to avoid the rather passive situation in which the students just fill in a given report structure. On the contrary, defining the best structure for the report is an important part of the task (and it is generally the case in real like). The teachers are fully available to discuss the proposed structure with the students, as well as any other aspect of the assignment. Some of the "TP classes" are specifically devoted to provide such support to students.

4. Resources

Students are supposed to use their own computers, including virtual machines in the systems infrastructure of the Department. Obviously, the machine where the web service is running must have a public IP address and the corresponding port open, so the other student groups can use the web service.

The following supporting material is also provided together with the present description of the assignment:

- A mini-tutorial on web services in Java and a simple demo (zip file named ws_tutorial_demo).
- A javadoc describing the interface to be implemented by each group of students (zip file named javadoc_interface).

Obviously, the mini-tutorial and the javadoc assume the students will use Java. However, if the students prefer to use another programming language that is perfectly fine (actually, language diversity is recommended in N-Version programming).

5. Calendar and miscellaneous

The assignment must be done by groups of up to 3 students according to the following calendar:

- March 23th – Send an email to the teachers (henrique@dei.uc.pt, rbarbosa@dei.uc.pt) with the names and emails of the students that constitute each group. Note that the group of students must remain the same for the forthcoming assignment 2.
- May 17th – Submit the code and written report (a zip file submitted at Nonio). Submissions after this date (May 17th at 23:55) are not possible/considered.

In addition to the oral defense that will cover both the assignment 1 and 2, the teachers will provide individual feedback in the results of this assignment.