DIT635 - Assignment 1: Quality and Unit Testing

Due Date: Sunday, February 16th, 23:59 (Via Canvas)

There are four questions, worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team (zipped archive) on Canvas. Answers must be original and not copied from online sources.

Cover Page: On the cover page of your assignment, include the name of the course, the date, your group name, and a list of your group members.

Peer Evaluation: All students must also submit a peer evaluation form. This is a seperate, individual submission on Canvas. Not submitting a peer evaluation will result in a penalty of five points on this assignment.

Problem 1 (12 Points)

The following properties emerge during the development of the software for an Automatic Teller Machine (ATM). For each of the following, identify whether it is a correctness, robustness, or safety property. Briefly justify your decision.

1. If the network connection is interrupted, the session shall be immediately terminated with an appropriate error message. No funds shall be dispensed, and no changes shall be made to a user's account.

2. The account identifier of the account loaded by the utility shall exactly match the account identifier of the input debit card.

3. The amount of money requested by the user shall only be debited from their account following a confirmation from the ATM of the successful withdrawal of physical funds from the machine.

4. If no physical money remains in the unit following a completed transaction, the utility shall cease standard operation and display an error message until a manual override is initiated.

Problem 2 (8 Points)

Under what circumstances can making a system more safe also make it less reliable? Briefly explain with an example.

Problem 3 (25 Points)

Consider the software for air-traffic control at an airport (say, GOT). Air traffic control (ATC) is a service provided by ground-based air traffic controllers (the users of this system) who direct aircraft on the ground and through controlled airspace with the help of the software. The purpose of this software is to prevent collisions, organize and expedite the flow of air traffic, and provide information and other support for pilots.

The software offers the following features:

- Monitors the location of all aircraft in a user's assigned airspace.
- Communication with the pilots by radio.
- Generation of routes for individual aircraft, intended to prevent collisions.
- Scheduling of takeoff for planes, intended to prevent potential collisions.
- Alerts of potential collisions based on current bearing of all aircraft.
    - To prevent collisions, ATC applies a set of traffic separation rules, which ensure each aircraft maintains a minimum amount of empty space around it at all times.
    - The route advice can be either of "mandatory" priority (to prevent an imminent collision, pilots should follow this command unless there is a good reason not to) or "advisory" priority (this advice is likely to result in a safe route, but a pilot can choose to ignore it).

You may add additional features or make decisions on how these features are implemented, as long as they fit the overall purpose of the system. In any case, state any assumptions that you make.

For this air traffic control system:

- Identify one reliability (not assessed using availability), one performance, one availability, one scalability, and one security requirement that you think would be necessary for this software.
- Develop a quality scenario for each to assess whether the final air traffic control system will fulfill the desired quality attribute.

State any assumptions you make about the design or functionality of this software.

Use the quality scenario format from Lecture 2: Overview, System State, System Environment, External Stimulus, Requires System Response, Response Measure

Requirements should be specific and testable. Scenarios should have single stimuli and specific and measurable system responses.

Problem 4 (55 Points)

Software engineers love caffeine, so we are planning to install a new coffee maker in the classroom. Fortunately, the CSC department at North Carolina State University (NCSU) has developed control software for a shiny new CoffeeMaker, and has provided us with that code. We just have to test it.

You will be working with the JUnit testing framework to create unit test cases, find bugs and fix the CoffeeMaker code from NCSU's OpenSeminar project repository (thanks to the authors!). The example code comes with some seeded faults. For this exercise you are required to create a test plan, implement the test plan as unit test cases for all the application classes with JUnit, execute those against the code, detect faults, and fix as many issues as possible.

Your submission should include:

1) Test Descriptions. Based on your exploration of the system and its functionality, you will formulate a plan for how you will write unit tests for the application classes to ensure that they deliver system functionality, free of faults. This document will describe the unit tests that you have created, including a description of what each test is intended to do and how it serves a purpose in verifying system functionality. Your tests must cover the major system functionality, including both normal usage and erroneous input. You must explain why your tests will cover all major functionality, and not leave gaps in test coverage.
    a) As the Main class requires keyboard input, you do not need to write unit tests for it. Instead, your unit tests should target the other classes of the system. However, we would recommend performing some exploratory testing (human-driven exploration) using the Main class to understand how the system functionality should function before you write unit tests.
2) Unit tests implemented in the JUnit framework
3) Instructions on how to set-up and execute your tests (if you used any external libraries other than JUnit itself, or did anything non-obvious when creating your unit tests).
4) List of faults found, along with a recommended fix for each and a list of test cases that expose the fault.
5) If you find faults (by other means, such as exploratory testing) that are not detected by your unit tests, describe the additional unit tests that you would need to detect those faults.

Relevant links:

- CoffeeMaker code and sample tests **(do not turn in the sample tests as your own)** - http://Greg4cr.github.io/courses/spring20dit635/Assignments/CoffeeMaker_JUnit.zip
- jUnit - http://junit.org/
  (We recommend using a Java IDE - such as Eclipse or IntelliJ - that makes it easier to integrate JUnit into the development environment.)

- IntelliJ - help on setting up JUnit - https://www.jetbrains.com/help/idea/configuring-testing-libraries.html
- Eclipse - help setting up JUnit - https://help.eclipse.org/2019-12/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm

Points will be divided up as follows: 15 points for test plan, 20 points for unit tests, 10 points for detecting faults, and 10 points for the suggested fixes to the codes.