**Project 2**

Zachary Mohler

Southern New Hampshire University

CS320

Linda Gaston

August 9, 2022

For the development of *Grand Strand Systems’* application, I was responsible for the creation of three features that would appear in the final application. These features were the *task service, contact service,* and *contact service* features.

My approach to development of all three features was largely the same seeing as they are very similar in implementation. This also led to a very similar testing approach which ultimately yielded similar test results which meet industry standards for completeness. My approach in development aligned directly with the requirements outlined. For example, I would first create any classes required by the program, I would then create any required data elements, and finally I would build the logic of methods required for requirements to be functional. These methods were only roughly described by the requirements, such as “add” or “remove” element from the service class. This prompted me to have to design my own implementation, and therefore create some supplementary methods that weren’t described by the original requirements. Some such methods were *getId(), getName(), etc.* Ultimately some basic getters and setters to make it possible to access and identify certain elements. These supplementary methods were completely necessary for the program to function as per the requirements and, in order to keep complexity and storage to a minimum, were only added when completely necessary.

My testing procedure, as previously mentioned, followed the same structure for all three features. The first stage of testing was working tests, that is, testing during development. Working tests not only ensure the code is syntactically error free, but also ensures implemented features behave in a way that is to be expected on the surface level. It should be noted that I do not thoroughly test logic at this stage, the testing conducted is essentially just to ensure the functions can be called. After the development stage is completed and I had a working product, I began the process of creating a JUnit file to thoroughly test the logic of my implementations.

My approach to writing JUnit test cases was very similar to my approach to the requirements-based development I performed. I would build test cases based upon requirements. For example, in the contact service feature, there is a requirement to be able to change a contact’s name by supplying a contact id and a new name. To test that my implementation correctly addressed this requirement, I would first initialize a contact instance and add it to a contact service instance, I would then test to ensure that the old name had properly been added to the contact, after that I would make the change and test again, this time ensuring the new name had been assigned to the contact. This way, I had properly demonstrated that the initialization of the contact worked, as well as the change name method.

Another way I ensured my JUnit tests properly testing the code base was the use of Eclipse’s JUnit code coverage. What code coverage illustrates is the number of lines from the implementation that are ‘touched’ by the test case, that is, lines that are run when testing. Code coverage also considers different branches of the code. For example, when an if statement has two possible branches, code coverage considers each branch its own line so if you only test one branch, that wouldn’t be considered a complete test. The code coverage percentage is calculated by dividing the total number of instructions by the total number of instructions touched by the test case. For my implementations of the features, I achieved an average code coverage of 83.2%. A generally accepted code coverage is 80% (Pittet, 2022).

One way I assured my code was technically sound was to test all possible error cases, as well as confirming successful cases worked properly. For example, in the test file for the contact class (ContactTest), to ensure that a contact’s name data meets all requirements *as well as* correctly assigns the value when all requirements are met. I first test the initialization form of setting a contacts name, and test that it’s error cases handle the error properly:

Name too long: **Assertions.assertThrows(IllegalArgumentException.class, () -> {new Contact("ok", "01234567891", "ok", "okphonelen", "ok");});**

Name null: **Assertions.assertThrows(IllegalArgumentException.class, () -> {new Contact("ok", null, "ok", "okphonelen", "ok");});**

Then, I test that the assigned value is assigned when valid input is received:

**Contact toTest = new Contact("44", "Jerry", "West", "3235550010", "3900 W Manchester Blvd");**

**assertTrue(toTest.getFirstName().equals("Jerry"));**

After the initialization test, I must run the same tests within the setter scope to ensure when the setter is called the same requirements are met.

To ensure my code was efficient, I employed industry best practices for data structures when designing my implementation. The data type I decided to use to store elements in the service instances was an *Array List.* This may seem inefficient at first glance due to the linear efficiency of traversing an array list—but given the context of the application it is safe to assume that there will never be more than a few hundred entries in any of the service features. In addition, if the program scope changes in a way that requires a change of data type for storing contacts—my implementation lends itself well to that sort of change. The data type could be easily changed to a hashmap with minimal changes to the code base.

In conclusion, I believe the code coverage of my code base, along with the success of my test cases proves a satisfactory testing of my code base. I also believe I have illustrated my test cases cover the technical requirements of the application in an efficient manner.

The mindset I adopted when working on this project was strictly to address all requirements to the best of my ability while considering the security and needs of the end user. This made it greatly important to consider the complexity of the code, and in my case, minimizing it to the best of my abilities. For example, when traversing the array list of contacts, I could’ve traversed the array myself seeing as each element was an object I designed, but I ultimately felt it best to design my implementation to work in accordance with the existing array list methods to minimize the additional code I would write as well as keeping the complexity of my code base down.

On the topic of limiting bias, I found it very important to approach writing tests not as “confirming my code works”, but rather “finding holes in the logic of my implementation” after all, errors are almost certain to exist in every program. It is understandable that someone testing their own code would simply try to prove their code does what it is supposed to rather that stressing the program to find its inevitable weaknesses. I pride myself in limiting my bias when testing my code and try my best to find errors rather than confirm my success.

I have a very strong opinion on remaining disciplined in the commitment to creating robust, error free code. Developers should *never* allow themselves to cut corners when it could endanger the security or requirements of the end user. It is not only a matter of maintaining reputation, but an ethical issue as well. It has been shown many times before that erroneous code can lead to death such as in the case of the 2018 Uber self-driving vehicle incident (McFarland, 2020) where a cyclist was struck and killed by a self-driving vehicle due to a software issue. It is more important than ever to remain vigilant to create robust, error free software.

*References*

Pittet, S. (2022). Introduction to code coverage. Atlassian. Retrieved August 9, 2022, from <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage>

McFarland, M. (2020, September 18). *Uber self-driving car operator charged in pedestrian death*. CNN. Retrieved August 9, 2022, from <https://www.cnn.com/2020/09/18/cars/uber-vasquez-charged/index.html#:~:text=Rafaela%20Vasquez%20was%20watching%20television,National%20Transportation%20Safety%20Board%20investigation>.