**Project Two: Summary and Reflections Report**

Arica Bryant

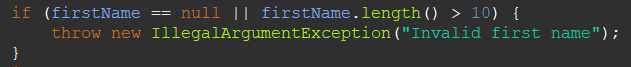
Southern New Hampshire University

CS-320 Software Test Automation and QA

Professor Jonathan Norman

**Summary and Reflections Report**

For the milestone assignments, I ensured that my Junit tests were aligned with the software requirements. For example, for the Contact Class, one of the software requirements was “The contact object shall have a required firstName String field that cannot be longer than 10 characters. The firstName field shall not be null.” Based on these requirements I knew that I had to test to make sure the firstName string was not longer than 10 characters. I also had to create tests that confirmed the field was not null. I was able to achieve this by setting up testing conditions that threw exceptions in my Contact Class:



Then I created Junit tests in my ContactTest Class to test the functionality and catch exceptions:

A screen shot of a computer code

Description automatically generated

To determine whether my Junit tests for the contact and task service classes were effective, I used coverage testing and monitored my percentages for the Contact and ContactService classes. A high percentage above at least 80% is a good amount of coverage. My Contact Class has a percentage of 87.2% and my ContactService Class has 100.0% coverage.

A black background with white text

Description automatically generated

After working with the Contact classes, I was able to improve my percentage for the Task classes. My Task Class has a percentage of 100.0% and my Taskservice Class has a 100.0%

A black background with white text

Description automatically generated

After some code refactoring and all three milestones were combined into one project, my overall coverage percentage was 97%.

I ensured my code was technically sound by providing comments throughout the program. Specifically, comments before each test case, explaining what the particular section of code was testing. I also tried to make sure my variable names were clear and easy to read. This improved the readability of my program.

When working to make my code efficient, I tried to make sure I did not reuse code. In my Contact Class milestone, I repeated my validation code in both a constructor and the setter methods. For my Task milestone, I created validator methods that I could repeatedly call when needed instead of retyping code each time.

A computer screen shot of text

Description automatically generated

A screen shot of a computer code

Description automatically generated

I utilized white-box testing techniques throughout the milestone assignments. White-box testing focuses on creating tests based on the code developed to implement a component or system (Hambling et al., 2019). In this case, I used JUnit testing as a form of white-box testing. JUnit tests were employed to ensure that the program functioned according to the software requirements.

To begin the testing process, I reviewed the rubric and made notes on each requirement. These notes were then transformed into JUnit tests and systematically added to the code one at a time. Before I moved on to the next method, each one was thoroughly tested for complete functionality. I made sure that the method section was highlighted in green before moving on. To ensure good testing coverage overall, I aimed for a JUnit coverage percentage of above 80 percent. In addition to JUnit testing, I also incorporated regression testing into my program. This involved testing after any changes were made to the code to catch any unintended effects on the existing functionality.

There are some aspects of white-box testing that I did not use when working on the milestones. I did not fully understand the overall purpose of the project and I only knew how the specific sections should function. Despite this, I just focused on testing the specific aspects of the code based on the requirements. White-box testing, specifically JUnit testing, was very valuable for detecting bugs and defects early in the software development process. By testing each section of the code independently, I did not need to know the overall function of the program. Regression testing is also useful for any software development project because it is good to test frequently and often to ensure not too many bugs occur late into development, costing both time and money.

When working on this project, I realized how important it is to test code thoroughly. I exercised caution by only adding and individually testing one section of code at a time. This ensured I did not fill my program up with untested code that could break and be difficult to fix. I know that there may be bias in my test process because I am responsible for testing my own code. I stayed on track by sticking to the specific requirements and testing based on the information given to me.

There are many ways that I can use to prove my testing skills, and this begins with having an understanding of how important testing is and why it is done. I never want to cut corners when it comes to software development. The consequences of rushed, errant code can be extremely detrimental, and people can be hurt. Regression testing should always be used during code refactoring. I will work to avoid technical debt by testing early and often throughout the development of the project. I would work to avoid bias by also allowing others to test and analyze my code. I will also work to create and update documentation for the project to ensure development is staying on the right path and the project is maintaining its original functionality.

**References**

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.