**Reflections Report**

1. **Summary**

![Graphical user interface

Description automatically generated with medium confidence]()When it came to my unit testing approach for the task, I found that performing white-box techniques for a majority of the task to be extremely beneficial. This was mostly done by ensuring that tasks such as the ID being no longer than ten digits or making sure that the contact first and last name are no longer than 10 characters each. These tasks would be checked by creating “IllegalArgumentExceptions” that would essentially attempt to pass an illegal statement only for it to get caught and thrown. After it was caught and thrown, an error message would appear for the user to know that the attempted input is invalid.

 Additionally, I made assertions in the program that would assert an object of the task to possess illegal arguments for the program to detect and throw, such as a field being null.

With a coverage of about 81.0% and the task requiring that the coverage be about 80%, this is one of the reasons that I know my JUnit tests were efficient.

Writing out my JUnit tests was a different experience than I was used to. In writing the JUnit tests, I found that I was following a checklist of what needed to be tested in regard to functionality and reviewing the syntax of the code rather than it’s behavior through execution. This static method was new to me, but not something that was too challenging to adapt to. Verifying that all of the tests passed helped in verifying that the code I was producing was ![Graphical user interface, application

Description automatically generated]()effective.

1. **Reflection**

Some of the software techniques that were employed in the creation of this project included both static and dynamic testing with some white-box methods being implemented as well. Static testing is a testing method that involves reviewing your code without necessarily executing it (Hamilton, 2023). This form of testing is more or less done so to fix any errors that could exist with the flow of the program, structure of the software, or syntax errors caught by the verification checker often found in IDE’s or the programmer.

Dynamic testing involves running the code in order to see how it acts by observing the behavior (Hamilton, 2023). Since there is no user interface for me to interact with, the “behavior” being observed would be if errors would be thrown or if the JUnit tests passed as intended. According to Hambling, “… white-box tests focus on ensuring that particular elements of the structure itself are correctly exercised.” (Hambling et al., 2019). This was performed by working on each of the components individually and verifying that all the tests would pass before moving onto the next (Contact.java/ContactTest.java first, then ContactService.java/ ContactServiceTest.java for example) to make sure it all worked properly. Coverage tests are also involved in this style of testing to discover how much of the code if being looked at as the program executes. Methods that weren’t included consisted mostly of experience based techniques as I’m still fairly used to the process. The only one I believe I implemented could have been checklist based testing, but even that is a stretch. Other not included testing forms are techniques involving performance, security, and any other non-functional requirements testing.

As the tester, it’s important to appreciate the complexity and interrelationships of the code so that it can be easier to identify where potential problems could or have occurred. The extent of caution that was implemented was by considering all the tests that could be performed while staying within the limits of the functional requirements. Understanding the tests that have to be performed can also help decide what code should be used in order to accomplish the task.

When it came to eliminating bias in my code, the best method I had was by attempting to solve it as objectively as possible. That being said, being strict on my own code was one of the tactics I used to eliminate this bias. It’s important to be critical of your own code, especially when trying to remove your program of as many defects as possible. I would make sure to verify that each test would pass at least five times in a row to ensure that my code was efficient and not just lucky.

Being disciplined in the commitment of quality as a software engineer is essential if you want to become successful let alone a professional in the craft. As a professional software engineer, it would be safe to assume that the software that is being worked with impacts the lives of hundreds if not thousands or more in one way or another. Whether it’s software for convenience or for saving lives, it’s important to never cut corners in testing code since we want to ensure that the customer is obtaining the best product possible with as little defects in it as possible. Customer satisfaction is always something to consider, but the safety of the customer(s) and their information should also be considered first and foremost. Taking the time to test the product thoroughly, while possibly being costly in that stage, will be far cheaper than releasing a product to only have it be re-called, re-serviced, re-tested, then re-released to the public.

**Resources:**

Hambling, B., Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). 4 Test Techniques. In *Software testing an ISTQB-BCS certified tester foundation guide - 4th edition*. essay, BCS, The Chartered Institute for IT.

Hamilton, T. (2023, February 14). *Static vs dynamic testing: Difference between them*. Guru99. Retrieved April 13, 2023, from https://www.guru99.com/static-dynamic-testing.html

Hamilton, T. (2023, March 11). *What is dynamic testing? types, Techniques & Example*. Guru99. Retrieved April 13, 2023, from https://www.guru99.com/dynamic-testing.html