**Jared Flynn**

**CS 320 Module 7-2: Project Two**

**10 December 2023**

**Summary and Reflection**

**Summary**

My testing approach was in complete alignment with the software requirements as there was a test provided for each specified requirement. The contact class had requirements to input a first and last name with a maximum character length of 10 and cannot be null. The Junit test checked if the input for the contact’s name was too long or if it was empty and would through an exception if this was the case. The task class required a unique task ID to be generated with a max character length of 10 and could also not be null. The Junit test checked whether the ID was too long or empty and whether it was a duplicate or unique ID. If it did not meet any of these conditions, it would throw an exception.

To ensure my code was technically sound I used strings and a list array for each of the different classes. To ensure the contacts were properly created and stored I used a list array with the following code:

public static List<Contact> contactList = new ArrayList<Contact>();

TO ensure my code was efficient I periodically reviewed it looking for ways to make it simpler and reduce any redundant lines of code. Generally, I refer to external sources like YouTube, Reddit, or StackExchange for different ways to make my code more efficient. When creating the code to test the contact class I initially had an excess number of lines that essentially accomplished the same thing of creating a contact and testing a specific method. Initially I had manually input the test strings into each test function however I changed the code to set my test strings as variables and utilized a single function that referenced those variables instead and was able to make it much cleaner and much more efficient.

**Reflection**

The primary testing technique I employed during this project was unit testing through JUnit testing which is used to test code to ensure it functions properly. It does this by breaking it down into small chunks checking it for errors and if it meets the conditions before going to the next test. Additionally, I used clear naming conventions to identify the test methods with those of the methods being tested. I also employed static testing to try to identify any errors before I ran the code. This helps to ensure there are not any blatantly obvious errors before running the code.

Software testing techniques I did not employ would be non-functional testing which tests non-functional requirements such as security usability and reliability. Acceptance testing was also not included as this is the last phase of software testing to ensure the system is ready for actual use and since this is for a school assignment and not an actual product there was no reason to incorporate acceptance testing.

Each technique has its own practical use and implications. JUnit testing is a repeatable and automated code technique that ensures bug free code. Static testing allows for a more in depth approach to debugging code and can also be accomplished with many automated tools such as plugins for the many IDEs available for programming. Functional and non-functional testing is used to ensure software meets all applicable requirements while also being user safe, reliable, and user friendly.

My mindset while working on this project was to ensure I had efficient clean code that could be easily repeated. Each individual class functioned similarly to the previous meaning that the code could be reused and adapted for the next assignment. This made it easy to save time on programming and spend more time on developing the tests as well as debugging. I am still a novice when it comes to programming so understanding the interrelationships of the code is crucial to ensure I am not doing something incorrectly. This also led to a bias in that I was incorrect most of the time when developing my code. I am never certain if the way I am coding is the correct way although there is never truly a singular correct way, there are just more efficient ways. Insecurity is where my bias tends to lean.

Discipline is vital when writing code as cutting corners can lead to inefficient and buggy programs. This can also lead to security issues as one may overlook common issues that would otherwise be avoided by a more disciplined programmer. Ensuring our code is free of errors ensures confidential information is properly secured. Without disciplined programmers, vital services would not be able to trust the programs they need to use for industries like finance or health services. By following industry best practices and always looking for ways to improve my coding skills I plan to avoid technical debt.