The extent of my testing approach to the software requirements was for the application to respond correctly based on the requirements. The Contact ID, first and last name inputs must be ten characters or less and cannot be null, the Phone number field must be exactly ten characters, and the Address field must be thirty characters or less If all the JUnit tests come back one hundred percent, then the quality of the JUnit test will be of the intended quality. As I learn more about testing and the software development cycle, I will become more proficient in creating more intricate tests.

I try to always focus on having the simplest code possible, so unless absolutely necessary I will use a catch all approach or try eliminating the ability to make an error. Java exceptions are great because you can just send a general exception and describe all possible errors to the user.

1. private Task TaskSearch(String id) throws Exception {

2. int index = 0;

3. while (index < taskList.size()) {

4. if (id.equals(taskList.get(index).getTaskId())) {

5. return taskList.get(index);

6. }

7. index++;

8. }

9. throw new Exception("Task not found");

JUnit tests allow us to test small part of an application. These tests are independently compared with their expected results to ensure the actions of one component are not conflicting with another. In the contact and contact service classes we are comparing the class variable with input of the correct size, and incorrect size both short or longer than expected. This allowed me to see if the program will act appropriately when incorrect input is received as each class is ran. Using the assertEquals() method I can see if the value entered was the correct size by evaluating the Boolean when it returns false. The assertNotNull() method is also used to return if a input is empty or null.

This Document includes a follow-up summary and my reflections explaining how I analyzed various software testing for a customer’s application. The software testing was based on requirements supplied by our client to be applied appropriately so that the testing strategies met said requirements. The report includes highlights of testing techniques and mindsets adopted during the project’s duration.

I ensured that the code was efficient by again keeping it simple. I am not reinventing the wheel and I am not skilled enough yet to think too far outside of the box. My main goal is to have solve as many problems as I can with the time that I am given. Any of the assert methods demonstrate whether the class responds correctly to invalid input. There are no math formulas or other components to tests besides whether the input was the correct size based on the requirements which kept the Unit tests simple.

The majority of the testing we did would fall into white box testing. I have an idea of what parameters are going to be input and the language capabilities, and limitations of the system and its components. Under the umbrella of white box testing we utilize fault based tests and statements coverage tests. Several of the JUnit tests in the project test for faults like having empty variables. AssertNotNull methods asserts the actual value of the object passed to it is not null. If the object passed to it is null, then the method will return false. During the project we performed tests like statement coverage by testing incorrect and correct values. Using assert methods, the Junit tests checked for the correct response from the class functions. The AssertEquals method asserts that the actual value and expected value are equal. It will Pass if the two objects passed to it are equal otherwise it will fail. AssertNotEquals method asserts that the actual value and the expected values are not equal. If an object is passed that is equal to the expected value it will fail, otherwise it will pass. An exception could technically fall under coverage or faults testing. AssertThrows method asserts that expression will throw an exception of a type. It fails when no exception is thrown, or it is of a different type, for example an IllegalArgumentExceptions versus a NumberFormatException.

Technically Black box testing techniques were not used during most of the milestones. There was always a general idea of the system capabilities and functionality. Integration testing and system testing techniques were also not used for most of the milestones . I very lightly used integration testing when all of the classes are combined ensuring they each played well together. I eventually had had to perform system tests to see if it would perform all the necessary functions when everything was put together.

Simplicity is my goal for most projects and that did not change for this one. I didn’t assume the code was perfect, I even knew that users could enter invalid input. Keeping that in mind I only needed to know if the input values are valid based off the parameters. Exceptions and loops ensure that the values entered will meet the requirements so, focusing on if exceptions are caught was the highest priority.

I am not experienced enough in coding to have much of an ego but having read about a testers mindset I had a plan. I should assume there is an error and try to prove that. By keeping the requirements open I could focus on what needs to be tested rather than assuming it will be ok. Given enough time I will be more experienced and have bias but will also have peers that can help with testing or tools for testing larger projects.

Cutting corners can be tempting but usually causes errors further down the line. Although it may save time in the short term, undoing or correcting errors later can take longer than planning out a solution. Technical debt is the time it takes to rework software, as correcting code has a snowball effect on time to ensure the code functions correctly and as the client expects it to. I plan on avoiding the costs of technical debt by applying agile technique to my coding. Agile techniques involve frequent reflection and testing of code to ensure it meets the client needs. I create functions to display what is happening at different times when accessing data structure or performing critical operations. JUnit tests have become another tool that is critical in development as a Class’s Unit test can be made as we are writing the class to ensure it meets it requirements.

Works Cited

1. Debt, H., 2021. *How to Avoid Technical Debt*. [online] Full Scale. Available at: <https://fullscale.io/blog/avoid-technical-debt/> [Accessed 18 October 2021].
2. R., P., C., Gupta, L., A., Gupta, L., & R. (2021, August 26). *JUnit 5 Expected ExceptionassertThrows() Example*. HowToDoInJava. <https://howtodoinjava.com/junit5/expected-exception-example/>
3. Rana, K. (2021, May 28). *Difference Between System and Integration Testing*. ArtOfTesting. <https://artoftesting.com/difference-between-system-and-integration-testing>
4. S., Raj, A. P., & B. (2020, December 25). *JUnit 5 Assertions Examples*. HowToDoInJava. https://howtodoinjava.com/junit5/junit-5-assertions-examples/