### CS 320 Project 2

I developed three services, contact service, task service, and appointment service. Each service uses an object to perform various tasks and keep records of data. This data can be manipulated and changed depending on the requirement for each service. In addition, each service and object has a test for it that demonstrates correctness and stability, even when given incompatible inputs.

The testing approach aligned with the requirements in that each test that was written, was written to test a specific requirement. Both of the classes written contain their own set of requirements that must be met, so each class has a test case associated with it. An example of this would be how the contact and task classes have input requirements. A requirement for the task class was that the name of the task couldn’t be longer than 10 characters and couldn’t be null. Our test class verifies this by first creating and testing a working version of that object, and then creating a version of that object that generates the appropriate error if the requirements are not met. We use the methods testTask to build the task and testIdTooLong for the length requirement. This ensures that the requirements are all met while creating a clean and easy to use test case. Doing this for each requirement for each class confirms that the requirements of the project as a whole are being met. This also increases code coverage. Code coverage is a percentage of executed code during testing. Since each of my tests run each method, The code coverage is above 70 percent, and thus has a good code coverage percentage. There are available plugins for eclipse such as EclEmma, that measure coverage.

As stated before, each test covers the requirements for that specific class. Each of the requirements for these classes involve being able to successfully perform the functions required while maintaining data integrity. After running a function, I make sure to test each field to ensure maximum accuracy. In addition, I added comments and followed standard coding practices, making my code sound. Using the command *assertTrue,* a streamlined JUnit function, I was able to make the most of JUnit and create clean code. One problem that I experienced was that the way I had the tests set up, they had to run in a specific order. I realized that each test unit should be able to run independently of the other and in any order. By restructuring the tests to be self containing, I made the code more efficient. In the method testDeleteContact, I use the addContact method to add a contact that can then be deleted.

There are many different techniques and practices that can be used to ensure that the tests you create are efficient and effective. Because testing relies on a sample of data, it's important to make sure to test correctly or the results won’t be indicative of the system state.

One such technique is creating tests to be simple. Creating simple tests ensures that the tests can be easily viewed and verified by other testers. It also allows for the test to be adapted easily for future testes.This practice can be used in many ways to save money and time for any type of project that involves testing. In my code, I created tests that only tested one aspect of the requirements. Each requirement, excluding the id uniqueness requirement, has its own testing block, making use of the single responsibility principle. This makes it simple to understand where tests failed and what needs to be fixed. It also allowed me to adapt it to other projects. The delete test for all three services are very similar because they were adapted from one simple delete test.

Another principle of writing tests is the importance of implementing tests for both positive and negative results. Implementing tests for positive results shows that the system works as intended when it receives the intended input, but oftentimes things can go wrong if the wrong input is received. That's why catching and dealing with these inputs with negative testing is important. A practical example would be if an online shopper tries to put items in their cart that the store no longer has access to, or a quantity that isn’t available. Negative testing shows how the results of said inputs would be handled by the system, whether it be with an error code or a system crash. I used negative input for all of the objects I created for my services. Each input is tested and displays an error as intended.

When writing code, oftentimes best practices allow that code to be changed and adapted to fit other needs. In my code, I adapted many of the JUnit tests from class to another. I employed caution by looking over the code and comments to make sure there were no errors and that it was adapted properly to the new class. This is clear in my code as all of the test cases have similar structure and function, creating an overall more stable and standardized code base.

To limit bias in my code review I tested each class the same, making sure to not skip any potential inputs and functions in my review. This project has many different inputs and functions that have many different requirements, but I handled each case with its own test. I think that one of the biggest biases developers can have with their code is that the input will always be appropriate for the class it’s given too. I took extra care to make sure that all of my programs were tested for bad input handling and passed.

When writing code it’s important to remain diligent and disciplined. Cutting corners when developing leads to technical debt by creating problems later in development. It’s important to understand that everything being developed and created may not be completely understood or handled in the best way and may require reformatting. The first test case I wrote had many issues and errors and would have amassed lots of technical debt by creating functionality issues later during development. I often went back to refactor and rearrange my code to be more uniform and stable and overall complete for the system.

Altogether my code and test cases function as intended and can show that through JUnit. Automated testing with JUnit has been a very straightforward and systematic process to follow. By making sure requirements such as code coverage and efficiency are met, my tests are well made and show that the code is also meeting the requirements I set out to meet during initial development. As I completed each test, the next one became easier and I had more tools and lines of code that could be used for the next one. Altogether, automated testing is an efficient and clean way to demonstrate correctness, and find errors in software development.

Letouzey, J., Coq, T., Grant, T., & Sturtevant, D. (2016, February 28). Introduction to the technical debt concept: Agile alliance. Retrieved February 21, 2021, from https://www.agilealliance.org/introduction-to-the-technical-debt-concept/