**Summary and Reflections Report- Tom Czubat**

**Software testing is a vital part of the software development life cycle. Testing needs to be looked at be a looked at as an integral part of developing software and not an afterthought. It should be done throughout the development process. During this phase, requirements will be emphasized and tested for. All defects must be found and fixed to ensure the software is secure and is working properly. There were several different kinds of testing techniques that can be used. There are two different types of testing types which are static testing and dynamic testing. The difference between these two are that static testing is done without running the code and dynamic testing is done when running the code. In general, you would always conduct static testing before dynamic testing, so it is important to review your tests before running them. The earlier that errors are found, the better, and there will be less technical debt to deal with in the future.**

**I used several unit testing techniques when completing the ContactService, AppointmentService, and TaskService features. One technique that I used was boundary testing which was to make sure that the expected value was between the two extreme ends of the boundary. To use this, I had to know what would be the minimum and maximum value for the upper and lower boundary. This was used in all three of the sections to measure the amount of characters of certain strings such as the IDs or description length. When I would declare these variables, I would create separate variable to keep track of their max length so I could test for them. For example, I knew that one of the requirements was for the appointment id length to not be more than 10 characters so I would use a separate variable to make sure it was going over that boundary and this is shown below.**

**final** **private** **int** APPOINTMENT\_ID\_LENGTH = 10;

**Another technique I used was decision table testing which tests conditions after getting input from the user. This was used in my tests to validate the input from the user. For example, my TaskService class has a setName() method that would be called in the constructor that would check if the input was null or if it the number of characters in the name was greater than 20. If either of these conditions were true, an illegal argument exception would be thrown, and an error message would inform the user of their invalid input. The test coverage helped me in these test cases because I could figure out how effective and efficient my tests were. They heled me refactor my code and make it better. For example, when I was writing he TaskServiceTest file, I had incorrectly done my set-up phase because I had set the id set more than 10 characters which was not working correctly when I was making sure that the number of characters were 10 or less. I also had some trouble when it came to setting the date in the beginning because I had not accounted for making sure that it was not null. I think that my approach taught me that you need to focus on the requirements and handle them one by and to make sure they work before you move on to the next. The requirements are the most important part because they will let you know if your code is doing what it is supposed to do. I think it also helped to combine the testing for requirements into one test because this made sense in several cases because they go together. This is where the assertAll() function was big help because I was able to group similar tests and requirements together. This was used in my ContactTest class as shown below.**

@Test

**void** contactTest() {

Contact contact = **new** Contact();

assertAll("Constructor",

() -> assertNotNull(contact.getContactId()),

() -> assertNotNull(contact.getFirstName()),

() -> assertNotNull(contact.getLastName()),

() -> assertNotNull(contact.getPhoneNumber()),

() -> assertNotNull(contact.getAddress()));}

Combining tests made sense in this situation because it made the code my code more concise, and I did not have to write extra code. With this, I could check if the contactId, firstName, lastName, phoneNumber, and address are not null.

Some other techniques that I learned about was regression testing. Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. This can relate to my code since whenever I would write a new test, I would run the rest of the code as well to make sure the test I wrote did not have any unintended consequences. I think this is a really good testing method when you have a lot of files because if you use regression testing each time you create a new piece of code, it will be easier for you to find errors and fix them. If you do not regularly run your code, it could take you a long time to sift through your files to find the error. I think that regression testing allows you to be cautious and not break your code. I believe that there are two important objectives that the team needs to keep in mind when creating tests and those are requirements and simplicity. I think that tests should be simple, so they are easy to keep track off. When grouping tests together, they should be grouped into similar groups like when you are checking to make sure none of the fields are null. I also think that the requirements should remain a focus throughout the testing process. They should be clearly defined at the beginning of the process, so the testers know what they need to implement in their test cases. In this project, the testing revolved around testing the user inputs. I had to make sure that the tests cases accounted for null values and the input fields were not too long. This was the case when making sure the address field was no longer than 30 characters. Checking for equality was a main focus in the AppointmentService class. This is because I had to make sure I could add, delete, and update appointments so it was important to make sure that two ids were not equal and that was done using this line of code.

*assertNotEquals*(firstId, secondId);

-Working with tests during this semester, has changed my point of view one topic. At the beginning of the semester, I thought it might be possible for a developer to also be a tester but going through the different modules I have realized how much better it would be to have a dedicated testing engineer or testing team. These individuals can have a higher level of skill if all they focus on is testing. There is also the advantage of having a fresh set of eyes on the code to find any errors. One word that comes to mind when testing is completeness or coverage. I like seeing the coverage percentage because it reminds you that your test cases can be better can causes you to do some critical thinking and make sure you did not miss anything in your tests. There were many examples of this in the project such as making sure there was proper input validation by making sure there were no null values and making sure certain fields were the right amount of characters like making sure the phone numbers were exactly 10 digits. I believe that quality code is important and ensuring the quality of your code essential in developing stable software. Something that I will bring with me in the future is attention to detail and making sure that every requirement is tested for. Reading about the different incidence around the world that cost companies millions if not billions of dollars because their software was not tested properly has shown me how important it is to check the quality of your code and realize that just because it says there are no errors doesn’t mean that there are no defects in the code. Running these dynamic tests in a real-life environment is a must to make sure the software is of the highest quality.