Summary and Reflection Report

My testing approach was specifically aligned with software requirements. In both the milestones we had to have a certain amount of characters available for an entry and the entry couldn’t be null either. For example, the task name could not be greater than 20 characters or null. Each test took into consideration these restrictions to make sure encapsulation of these restrictions carried over. The restrictions were made during the set method originally. When I created the updateTaskName method, I tested for these restrictions and made sure that the throws occurred when an invalid name was input. At first, I was adding these restrictions to too many methods and with a little advice I realized that it wasn’t necessary to do so. I know my JUnit tests were effective because I made sure than when I ran them the coverage for the file being tested was covered at 100%. I didn’t stop writing JUnit tests until I achieved that. I did so for both the task service and contact service projects.

I ensured that my code was technically sound by making sure the code was working and passing the tests but also by following advice from the previous milestone. For example, I used local memory to create a unique ID in the Contact Service project but in the Task Service project I used singleton method to get a unique ID that was set into the memory and attained directly from memory as seen in Figure 1. All tests passed and values were added to the HashMap successfully. I ensured my code is efficient by removing redundant code and reusing components whenever possible. When I first created my code, it contained verification of valid input in a lot of the methods. I didn’t have to recreate this since the set methods I created already had this information in there and the throw value would activate whenever an invalid input was made. I deleted a lot of these checks and it condensed my code significantly. I also reused multiple methods throughout the code as well as to not have to rewrite the code again. In Figure 2 you can see how I reused the setTaskName method from the Task.java file in the TaskService.java file. I didn’t have to rewrite the set method and the validity checks were still in place as well to make sure Text

Description automatically generatedthat the input was an accepted value.

Figure 1

Graphical user interface, text, application, email

Description automatically generated

Figure 2

Figure 3 and 4 below show the tests that were created on the previous images to make sure all checks were in place and the assignments worked as intended.

Text, letter

Description automatically generated

Figure 3

Graphical user interface, text, application

Description automatically generated

Figure 4

Throughout this project, I used unit testing as my technique. This type of testing technique would be classified as white box testing or structure-based testing. The technique I used falls under this description because I have knowledge of the internal structure of the code. There was also a black box testing technique used as well. I created a class first, such as the Task class, then I wrote the code based on the requirements given. For this class we needed to have a unique ID that cannot be null or over 10 characters, a task name that cannot be null or longer than 20 characters, and a task description that cannot be null or over 50 characters. Since we will be testing these parameters, the unit tests used boundary value testing technique. The tests made sure that the input was within the valid character limit. Building the set methods to account for these I tested these parameters when creating the unit test. The constructor for the Task was made to call for the set methods. I first tested the constructor by passing the value directly into constructor and asserting value was true to watch I passed. Afterwards I tested the set method by creating a constructor then setting each parameter to a new value using set method. I tested that each value was now equal to what the set value was and tested the throw values of the set methods. I tested the throw values by setting the parameters over the character limits and by setting them to a null value. This made sure that the throw values were in effect. This type of testing technique would be statement coverage, a white box technique, since I made sure that all the functionalities are working at least once (Khandelwal, 2019).

Some other testing techniques that were not used would be security testing and state transition testing. Security testing is more like a process because it comes with a lot of internal steps to complete but it verifies and rectifies any kind of unauthorized access to the system as to help in avoiding any kind of breach because of hacking or cracking practices (Khandelwal, 2019). This type of testing was not performed because the system wasn’t built with security in mind, at least not at this stage. The program so far has been about entering information into a database or local memory, but it did not secure this information. Going further there would be steps put in place to protect user information but not currently. State transition testing is testing the software against the sequence of transitions or event among the test data (ReQtest, 2019). This type of testing was not used because there was no transition of data. The data was compiled but there was nothing done with it after it was saved. When we use up our attempts at entering a password and we are sent to another page stating we are locked out due to all attempts being used, that’s a transition. This code didn’t have anything of that sort. We only input and output data.

These techniques are considered either white-box, structural, or black-box techniques, functional. Unit testing, statement coverage and security testing are considered white-box techniques while boundary value testing and state transition testing are black-box techniques. Black-box testing is mainly used when the internal structure of the code is not known while in white-box testing it is. Testers are usually the ones who perform black box testing because they are given a code that they need to test for its function, and they don’t really need to know the bare bones of it. We can think of this when game testers get parts of the game to test and make sure that there are no game breaking bugs. White-box testing is the very first step of testing and it’s done by the developers themselves to make sure that they have working code. Both types of testing are essential in creating a working final project and both are necessary for the success of that project.

When considering the mindset adopted for this project, I’d have to say my mindset was based on knowing I must test the product which I am creating. If I had to choose a word for my mindset, I think it would be cautious because I haven’t had to test any of the code that I had written before, so this was new terrain for me. It was important to focus on complexity and interrelationships within the code because it made the code more concise and unique. As mentioned before, in the Contact Service milestone I originally used a local instance variable which was not unique if multiple service instances were created so instead I ended up employing a singleton pattern which could not be instantiated more than once and it constantly created a unique ID for each additional contact. To promote encapsulation, I also removed redundant validation checks throughout the code and only kept them in the set methods. Whenever the constructor was called, it called for the set methods, so any validation errors threw exceptions.

I believe that there was some bias in this project because I knew that whatever code I created would have to be tested by myself as well. Honestly, I can’t imagine any developer that wouldn’t have some bias when they develop and test their own code. The way I overcame bias is by welcoming instructor feedback, constantly making changes to my code to improve it, and always making sure that I tested the code thoroughly. The best way to remove a lot of bias is having another set of eyes on the code and I believe that having our instructor look at our code, give us advice and see our improvement going forward is what helped to remove some of that bias. The original code created in the Contact Service milestone and the finished code in the Appointment Service milestone was entirely different. It was so much more elegant and straightforward than the mess from the first milestone.

When starting this course, I would have to admit that I was intimidated by writing an entirely fresh code and testing it myself. In previous courses we had files given to us with some premade code and we just improved on it. It also did not help when our instructor made it very clear that we will fail if we plagiarize code. Not that I would but I feel like when I research how to do things I sometimes feel like I am copying their ideas but throughout this course I realized that I’m not actually doing that and instead learning how to incorporate their thoughts and ideas into my own and better understanding the subject. I believe this is where the discipline and commitment to writing my own code comes in. Using research to formulate my own ideas and creating code that is my own. For example, in the Appointment Service milestone we needed to create an appointment date field and it was advised that we use the Date class. I first created the code with that field, but I realized that I wanted to format the date a certain way. I also wanted to create the parameter to be a string input as well. After many hours of research and trial and error testing, I created the function we can see in Figure 5. I made the input for the set method be a string. The string is input in a mm/dd/yyyy format so I used the DateTimeFormatter class to format the date to what the string input would be. Instead of using the Date class I used the LocalDate class because I could parse the input string and the format to the LocalDate class. This created the object setDate and now I could set parameters to make sure that it can’t be null and to make sure that the date which is input won’t be before the current day. I didn’t just plagiarize the code; I persevered many hours of research and executions until I was able to formulate a working Text

Description automatically generatedmethod that had the required parameters.

Figure 5

This is how I plan to avoid technical debt as well. I don’t want to just take the easy way out and find a working code, I want to understand how to make my own code work, even if it takes hours and days. I didn’t expect this to be easy and I don’t plan to take the easy way out. In my opinion, software engineering is something that I’ll constantly be learning and no matter how much I will know in the future, there will still be more to learn. I really enjoyed these assignments throughout the course, and I enjoyed seeing improvement each time. The feeling of accomplishment I had when the code in Figure 5 ran and worked as I intended it to was amazing. I look forward to continuing my education in this subject and I cannot wait to start my career in this exciting field.

Work Cited

Khandelwal, A. (2019, November 16). 7 different types of white box testing techniques: White box testing tools. Retrieved February 21, 2021, from https://testinggenez.com/types-of-white-box-testing-techniques/

ReQtest. (2019, August 08). Black box testing technique, its types & approaches with example. Retrieved February 21, 2021, from https://reqtest.com/testing-blog/black-box-testing/