Braxton Morrow

Project 2

SNHU

CS 320

Prof Angel Cross

Due: 4/19/2022

Spring 2022

My unit testing approach was similar on all three assignments. I tried to group like objects close to each other in code, and then test each chunk with an, @Test. Looking at **Figures 1.1** and **1.2** (located on the figures page) which are TaskTest.java and ContactTest.java respectively, both from different assignments, I grouped appropriate objects gathered all the information I needed from them, and then before the lines of code are run an @Test is added at the top, to ensure that the entire block of code is running smoothly. Coding things in this manner allows me to better isolate blocks of code or in this instance different features that a program will accomplish, establishing which pieces may need work. This practice becomes incrementally important when working on much larger projects than this one, allowing for code scripts that are thousands of lines long, with many different chunks of code. Utilizing JUnit testing properly, testing and diagnosing large programs becomes a much easier task when using Java as your language of choice. I know my JUnit testing coverage was effective as it was written into every aspect of my programs, ensuring that each part was isolated as small as possible. Something I like to call a “Perfect Function”, in which a function only does one task as much as it needs to, then if more things need to be completed those get done in a different function, and then that first initial function gets called. I ensured proper testing coverage by testing each of these pieces individually. Ensuring each piece of my program puzzle was functioning properly. Looking at **Figure 1.3** which is lines 75 and 76 in ContactServiceTest.java, I utilized assertThrows() as sort of a try catch exception, as well as assertAll() in **Figure 1.2**, which does the same thing as assertThrows() but it ensures that everything passes, and if it doesn’t, gives a detailed report of what went wrong where.

The testing techniques I utilized in this project were a bit varied but primarily involved Static Testing which is what JUnit testing is. “Static testing involves manual or automated reviews of the documents… The main objective of static testing is to improve the quality of software applications by finding errors in early stages of software development process.” (Hamilton, 2022) Since I was not working in a team of any sort, I also used manual code review, to check my work ensuring that my code met specifications and functioned properly. Reviewing your own code make its much harder to catch mistakes, however if I can master this practice, I believe I can become an excellent software developer. Dynamic testing is another form of testing that I did not utilize, which means you actually execute the code you are testing. “The main objective of this testing is to confirm that the software product works in conformance with the business requirements.” (Hamilton, 2022). In other words, you want to use Dynamic Testing when you want to check the functional behavior of the system, executing the software and testing the output with the expected output. Like I said before I really tried to isolate functions as well as I could making each piece a part of the puzzle that was the entire system, and then testing each piece separately to ensure that each piece functioned as it should. As someone who was both the tester and the developer in this scenario it is almost impossible to completely eliminate all of the bias that is liable to happen. However if you approach each piece with the mindset of both, while still attempting to occasionally only think like one or the other at various points in the project you can try to simultaneously make decisions to benefit both tester and developer simultaneously, while still attempting to distance yourself from the possible bias that may occur. When it comes to writing and testing code, it is very important not to cut corners for a number of reasons. To start off we can talk about what I learned in this class, when it comes to testing at least making sure to do everything properly the first time is going to save you time later. I am a strong believer in never submitting a 2nd draft unless I absolutely have to, due to assignment or client requirements. If you do something right the first time it usually won’t need much to any at all in terms of improvement. Taking testing as an example, if I as the developer code something properly the first time, when I send the file off to someone on my testing team, or I test the software myself, as is the case with some companies, the only work the tester should really have to do is hit “Test”, and then submit a report stating that there were no major bugs or issues with the code I had submitted and he can go about his day. Be it logic errors, handling errors, actual code bugs, mistypes anything, if you do the job right the first time, you won’t need to go back and fix anything. Secondly, when it comes to prioritizing code quality, security becomes a major concern. If a developer was say working for a bank, or working for a coding firm that has clients that are financial institutions, you could lose them endless amounts of money simply because you forgot to insatiate something somewhere, and maybe that broke some sort of error catching function somewhere and now your client is getting huge amounts of SQL injections stealing thousands potentially millions of dollars’ worth of client data or assets. Ensuring your code is bug/error free is one of the most important things we as developers can keep in mind any time we sit down in front of an IDE.

**Figures**

Text

Description automatically generated**Figure 1.1**

Text

Description automatically generated

**Figure 1.2**



**Figure 1.3**

**Sources**

Hamilton, T. (2022, February 19). *Static testing vs Dynamic Testing: What's the difference?* Guru99. Retrieved April 15, 2022, from https://www.guru99.com/static-dynamic-testing.html