Matthew Taylor

7-2 Project Two Submission

20 April 2024

Summary:

In my project one testing, my approach was to break the requirements down as much as possible and then write one unit test per specification. I had some missing from module three, but one example of this is the requirement defined as “The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.” In this case, I wrote a unit test that asserts that the correct error is thrown when attempting to update the name field with 21 characters. I also wrote a separate test that asserts that the correct error is thrown when attempting to create a task without a name. And lastly, I have a separate test that ensure that if the requirements are met, the task is created correctly, and the value is stored correctly. I took this same approach with each requirement.

The coverage percentage is another important part of writing unit tests. It doesn’t really ensure that the requirements are met, but it does help to ensure that all of the written code is actually tested. When I ran all of my unit tests with coverage analysis, the only part that wasn’t covered was my setters. Originally in the module three portion of project one, there wasn’t any additional logic in my setters aside from the typical logic. Many people do not care if setters are tested, but since I added some additional logic to them in the module four and five parts of project one, I tested that additional logic. Everything else was covered.

In order to ensure that my code was technically sound, I made sure that my unit tests tested both the happy and sad paths. I tested to make sure that with good input, everything worked appropriately, and I tested to make sure that with bad input, the correct errors were thrown to handle those issues appropriately. This is evident in the testAddTask test which makes sure that the task is actually added verifying that the valid input works. Then there is a test that tries to add a task to the task list that does not have a unique ID, and that test asserts that the method returns false as it is supposed to.

I made sure that the code was efficient by testing to make sure that appropriate errors were thrown when input was invalid. This helps with the efficiency of the code, because as the code is expanded on, improper error handling can lead to terrible efficiency or even complete failure. Two good examples of this in my tests were the tests testNameTooLong and testDescriptionTooLong.

Reflection:

The main software testing technique that I used for each of the milestones was unit testing with Junit5. For module three, four, and five, I wrote individual test cases that would test the logic in each method in each of the classes that were part of the assignments. I wrote a test to make sure that the objects were created correctly, and that they could be added to the ArrayList correctly. I also wrote tests to make sure that each of the requirements defined in the assignment were met. This included testing for the correct errors being thrown in the case of bad input. The goal of this technique is to try to write tests to ensure that all of the code written in all of the smallest units work properly.

There are many techniques that I did not use for testing the software that I wrote in modules three, four, and five. One of those techniques is integration testing. Integration testing is meant to test modules that work together to produce some output. This testing technique will evaluate if the integrated modules meet the defined functional requirements set out ahead of time. Another testing technique that I did not use is system testing. The same way that integration testing will test the integration of smaller modules that have each been unit tested, system testing will test a full system of integrated modules to ensure that the system as a whole meets the functional requirements necessary for the project.

Each of these discussed software testing techniques are applicable in different types of projects. For instance, I used only unit testing for the milestones in this course because the size of the project was not such that there was any need for integration testing or system testing. As a project grows in size and complexity, and as there are more modules that need to be integrated to form a system, these additional testing techniques become necessary.

As I was writing the software and the tests for that software, I had to adopt the mindset of a tester. As a software tester, I have to be careful to make sure that the software is really running appropriately and that there are no errors. Any errors can lead to serious consequences for the business, and for the users of the software. Within reason, the goal was to test every single thing I could possibly test. However, as a software tester, there comes a point of diminishing returns where you have tested pretty much everything, and to continue to invest time in the testing process probably won’t really realize many results. There needs to be some limitation to the testing, but that point does not come until after a lot of very extensive and thorough testing. In order to test so thoroughly, the tester must appreciate and analyze the complexity and the interrelationships between all the different pieces of programmed logic. One small example of this is the need to understand the way in which the contact class and the contact service classes are meant to interact.

Bias can be a real concern if the software developer is testing his own code as I was in this assignment. One major struggle that I ran into was that I had a hard time thinking through everything that needed to be tested because I had already written the code in a way that made sense and was working for me. In my mind, every part of it was already working, so I only felt the need to test the parts that I was not totally sure about. One of those instances was that in creating the setters in modules four and five, it seemed so simple to me, that I didn’t think to test it until I ran the tests with coverage analysis and found that I hadn’t tested that logic.

I think that the quality of software is really important because at the very least low quality software can be a headache to use and will therefore probably not be used, and at worst, it can cause some real damage for businesses and users depending on the issue. For these reasons, I need to make sure that I am not cutting corners in the software development and testing of that software.