Hatcher Blair

CS-320 Software Test Automation

Project 2

6/17/2023

My testing approach aligned with the software requirements 1-to-1. This is because the tests are formed based on the requirements. For the software to do what it is supposed to do you need to test it against all those requirements. These tests need to not only test that the software does what it is supposed to do when an input is correct, but also how the software handles bad inputs and edge cases.

The tests that were written for the three services cover every scenario possible. These tests cover every function in the software for strings that are the maximum length allowed and strings that are longer than allowed. For the strings that have an exact requirement, a shorter string was also tested. These tests also cover making sure that all ID’s are unique. The tests were written to cover 100% of all cases that were possible. The one test that was missing was trying to enter an incomplete contact or task. However, there is only one constructor for each of those objects and so there was not a need for testing the constructor. If there had been multiple constructors, then there is a need for testing each constructor to ensure that constructor functions as expected.

To ensure that my code was technically sound I tested the tests by having them print the results of the functions to the console. This allowed me to verify that the code was executing as expected and that the functions were being called correctly. I did not do this when writing every test, but I did it for every function to ensure that it was being called correctly. To help efficiency and to help ensure it was technically sound I used a standard set of variables that were on the edge cases of the software. These variables were defined on lines 9-14 of the TaskTest class and 10-15 of the TaskServiceTest class. These variables ensured that there weren’t any typos in any of the variables being used for testing and allowed for standard strings to be used when testing the edge cases. This was much more efficient than typing out long strings for every test. I also made sure that every test was only testing one thing. The exception to this was the testTaskCreate test. This test was to ensure that a task was being created properly. However, it also tested the getters for the task class. This was really testing four different cases because there were 3 getters being tested along with the constructor for the task class. All other tests only tested one specific scenario which made each test quick and efficient as well as made troubleshooting quick and easy because you were able to identify exactly where a problem was occurring instantly.

For the tests completed over the last three modules I used unit testing. This is where you test each individual component of the code to ensure that it is working as intended. I used Junit to complete these tests, which is a tool that allows for automated testing. These tests were all functional tests that ensured that all requirements were being met. The process for running these tests was simple. First, you identify all the requirements of the software and their edge cases. Then you write test classes for each of the classes that you are trying to test. In these test classes you make sure that you identify all the requirements of that class and all the edge cases. For example, if one of the requirements was that the ID of each item was unique and exactly 10 characters, you would write 4 different tests to test the ID. You would have a test to verify that an input ID of 10 characters functioned as expected, one with an ID of 11 characters, one with an ID of 9 characters, and one with 2 ID’s that were the same. This verifies that the ID feature is functioning properly, that the ID can’t be less than 10 characters, that the ID can’t be more than 10 characters, and that the ID’s need to be unique. This meets all requirements and is an acceptable test for this feature.

Some of the testing methods I did not use to test these classes include, non-functional testing, manual testing, integration testing, system testing, and many more. Non-functional testing is where you test requirements such as performance, security, and usability. There were no performance or security requirements for these modules and so there was no testing that could be done on those items as there wasn’t a criterion that the software needed to meet. Usability testing also could not be done because there was not any output for these classes, these classes where all services that would be implemented into a UI or a command prompt later and were not usable on their own. The same can be said for manual testing, manual testing is where a tester manually inspects the code and manually inputs data to be tested. I wrote the code and so there is not much value in doing more reviewing further than my initial review as it takes a second set of eyes to gain more insight on how to improve the code most of the time, and as the software was not usable it was impossible for a tester to input data into the software at this stage of development. Integration testing is done when you have multiple different components of the software all coming together. There was no integration testing done for these modules because there was no integration happening. For each of these modules there were 2 classes being developed, the object and the service that handled that object. These two classes were written to be dependent on each other and so there was some integration between the two, but not enough to justify proper integration testing. Once more modules are connected, then there will be proper integration testing completed. System is like integration testing but implies that you are testing the entire system whereas integration testing does not. Again, because there was not an entire system to test system testing was not performed.

The mindset I developed over this project was to be skeptical of everything. When writing tests for a new class I wanted to make sure that there was no stone left unturned. Meaning that I wanted to make sure that every function was tested in every way that it could be used and all the ways that it wasn’t used. I started every task assuming that the tests were going to fail and made sure to question everything that I saw. While the code that I was testing had very little complexity, it did have some interrelationships. It was important to appreciate those interrelationships when writing tests because there can be instances where the functions work in the base class but do not when being referenced through another class. Because of this, it is important to test all functions in the base class and when being accessed from another class. When assessing my own code, I limited bias by following a set of standards for the functions written in the code and a standard naming convention. This allowed me to evaluate the code based off those standards and not accept any bias that I had towards my own code. This also ensured that the code was coherent across all classes.

Maintaining discipline is crucial in the software development world. This applies across all areas of development from naming conventions to having standard procedures for how things are to be implemented. This is especially important when you are writing code that is going to be scaled upon later in the development process. If you are to implement something the easy way instead of the proper way, then it is going to cause a lot of headaches later in the development process and someone is going to have to spend a lot of time refactoring that code. This causes longer lead times and more cost to the company. By implementing things properly and making sure that your design is scalable you can avoid technical debt in the long run. For example, I’m working on a game right now and there are several different types of countertops that the player can interact with. When implementing the first type of countertop it would have been easy to make the logic and design for just that countertop. However, then when it came to implementing more in the future, I would have had to repeat that same process for every countertop. Instead, I made a prefab design that contained the logic and design elements that were shared with every countertop. This allowed me to build off that prefab design for all the future countertops and saved a ton of time as well as reduced the chance for errors because the base design had already been tested and verified.