Alex Wells

CS 320

Ms. Kalysa Wilson

August 13, 2023

Project 2: Summary and Reflections Report

My testing approach aligned with the requirements by testing the edge cases that weren’t to be allowed and making sure the program throws an exception when that happens. I know that my tests were effective because they had high code coverage which means that a large amount of the written code was executed by the tests which gives me high confidence that it is working as intended. I also manually reviewed the coverage report to ensure that critical paths for the software were being tested and not missed and swept under the rug by the high code coverage.

To ensure that my code was technically sound I wrote tests to test against all the requirements and for some tests I even wrote helper functions to ensure that the right object was being returned by the code. An example of this can be found in my TaskServiceTest class.

*A computer screen shot of a black background with white text

Description automatically generated*

My code has many functions that require it to loop through the list of objects which is an easy place to get bogged down performance-wise if the list gets larger as the function has a time complexity of O(N). To ensure that O(N) is the worst case, I made sure to return and leave the loop when the correct object is found so it only has to go through the entire loop when the object is not found or it’s the last object in the list.

A screen shot of a computer program

Description automatically generated

Additionally, for the TaskService instead of having the program generate the taskID like I did for the contactID and appointmentID the user has to input the task ID and to ensure that it is the same it must compare it to existing Tasks in the list of tasks. Originally, I had tough to add each new taskID gotten from the user to a separate list, but that was increasing my space complexity and since the time complexity would be the same no matter which list I searched through for a match, I dropped that idea and just iterated through the list of tasks.

For this application I primarily employed the technique of unit testing and writing tests to specifications. This means that I created tests for each function in the code to ensure that it met the client’s requirements and that it was fully functional. This also entailed making sure that the test coverage percentage was sufficiently high because that is a good guideline about how much of your code is being executed by the test. This is also called white-box testing.

A technique I did not use is black-box testing, which is designing tests and outcomes from requirements alone and then putting code through the tester until it works rather than worrying about the implementation details as much as white-box testing does. I also did not use integration testing or systems testing techniques at all as that was outside the scope of the milestones, but those involve putting the smaller pieces of code together to create the full system and ensure that by integrating the pieces no new errors are created.

White-box testing is practical when you want high code coverage in your tests, however this means that white-box testing on a large complicated system can be time and therefore money intensive so smaller teams may not be able to use the technique efficiently. On the other hand, black-box testing is much easier to set up, only requiring the inputs and outputs, but since it doesn’t necessarily test every line of code and generally has less code coverage than white-box testing. For normal use of the system this is likely fine, however it makes the system more likely to produce errors or even fail completely if edge cases are introduced.

In developing these modules, I employed caution to a great extent. I was careful when adding code that it was compliant with the requirements and would pass the test cases. It’s important to appreciate the complexity of the code you’re testing so that you both understand it and understand that small changes in code can have large changes down the line if it’s not properly tested. A good example of this is if I were to modify my generateNextID functions in either the AppointmentService or ContactService classes I would need to ensure that what I change the function too doesn’t impede already existing IDs to avoid duplication and errors down the line.

I tried to limit my bias in testing my code as much as possible. I did this by writing tests to match the project requirements rather than matching the code I had written. When developing software, it’s important to remember that the functionality of the code and overall system is the most important thing and limit your bias towards your own code and your own approach when it comes time to test it. This can be a concern because you may be tempted to modify the unit test knowing how your code works to make it pass, rather than modify your code to ensure it passes the unit test.

It’s important to be disciplined and write well documented and well maintained code to keep your commitment to quality as a software engineer. This is important because you or your fellow engineers may have to work on the code in the future and doing it well the first time will prevent small issues from taking large amounts of time to fix. I plan to avoid tech debt by avoiding going for a hacky but quick solution and instead focusing on maintainable and modular solutions to problems even if it takes slightly longer. A good example of this is in a personal project of mine where I’m trying to use Java to program a Black Jack game. I could just hack it together in a few hours using basic data structures and some hacky math, but instead I’ve gone for a full object-oriented approach so that it’s easy to read and easy to add more functionality to it as I can only work on it sporadically.