Nick Basile

06/20/2023

CS-320-T5522

Project Two

Unit testing is vital to developing software in the present-day state of technology. With many systems integrated with each other, and multiple people using the same set of software at once, looking for errors and flaws and identifying and fixing them before they can go live is a huge priority. In my case, the unit testing I approached for all three milestones were very similar. I needed to identify what the specific requirements were for each unit, such as at name length limitation, or to check for a unique ID for each object.

For the Contact classes, the limitations on the ID, Names, and Description length requirements were shared with both the Appointment and Task classes. The Appointment classes were a tad bit different, as I had to verify the Date fields were both not before the current date, in addition to being Date objects. Finally, the Task classes were both simple, yet different as I needed to test and verify that the Task Service fields needed to be updated by only the Name or Description fields, instead of all three.

I approached all of the software requirements the same for all three different collection of classes, I created all three collections based on the requirements that were given. A screenshot of a computer

Description automatically generated with medium confidence

I feel the JUnit tests covered and tested appropriately for the requirements that were given for each Object of each Class. Testing to see that both the fields were properly updating, the objects being deleted, added, and everything else that were given as requirements to make sure that even if somehow a false value was given, it would throw an exception and not allow improper entry. My experience with writing the JUnit tests were varied at the start, then became much easier further along. Near the beginning I had a rough time looking into how to actually test with JUnit tests, but after the Appointment classes I had a much firmer understanding of what I needed to write and test against. However, regarding JUnit tests, and testing in general, I feel that in a vastly more complex environment I have a lot to learn still.

A picture containing text, screenshot, font, line

Description automatically generated

To test the code to see if it was technically sound, I not only ran practical tests with both the JUnit tests for the requirements, but also ran the class services to ensure that the class creation was without any issues as well. A screen shot of a computer code

Description automatically generated with low confidence

To ensure that my code was efficient, I made all four classes, class services and related tests structured identically, as well as wrote them very simply to reduce issues with errors due to overly complex functions and function calls.

A screenshot of a computer program

Description automatically generated with medium confidence  
A screenshot of a computer program

Description automatically generated with medium confidence

Some software techniques that I applied to this project were both dynamic and statis testing methods. Manually going over the software and looking at how each set of Classes compared to each other, and looking to see if there were any discrepancies was the first thing I did. Afterwards, I looked over to ensure that the requirements given were properly tested against, as when I did the dynamic testing if those were not properly tested against, they did not give any errors and therefore the requirements would not be fulfilled properly.

There are various other testing techniques that I did not use in this project, such as usability, security, and integration testing and so on. These types of tests were not needed, as a UI was not a requirement for the project, security did not need to be tested against as it was in a isolated environment, and integration was also not a necessity as it was not being integrated into a server or database as per the requirements given.

The practical uses of a few of the techniques that I have given above are more for larger companies, or alternatively online services that deal with a large number of users and data. Security testing for example, would be vital to a company such as a bank, or financial institution. As well as usability testing, to ensure that the end user finds the UI intuitive and easy to navigate and follow. Not following those testing steps could have massive consequences in larger software development projects.

The mindset that I adopted working on this project as a software tester was one of caution. Thinking about how each object, function, and class interacted with each other and how to properly test for and against it to ensure that the requirements were met without failures was a new experience for me. The complexity of something as simple as this, with only a few classes caused me to step back and think about how the code worked and interacted with itself.   
A picture containing text, font, line, screenshot

Description automatically generated

A larger, more complex software project I can foresee would be hard to grasp without visual aids breaking things down, as the way multiple parts of it all working together and how they could potentially fail would be hard to grasp without.

I believe that when dealing with your own code and requirements, bias can be a big impact in how you both conceptualize, base requirements around, and also implement your final product. As you may miss key, critical details that another person would be able to point out in a very simple manner, which may not be obvious to you. Examples of this in my life specifically are around literature, and English. What may make sense to one person, may not make sense to another. In software, requirements that have been met that you have set, may not be all the ones that are necessary to have a end product that is successful.

The importance of self-discipline is key to any career, engineering or otherwise. A software engineering professional is not exempt from this, as cutting corners is unacceptable and can lead to extreme losses of both financial gains in a company, and can even result in the death of the career that you have worked hard to achieve. Testing also falls under this category, as letting code that has bugs, errors, or loopholes can potentially cause issues for end users, and even worse when systems do not work correctly.

Examples of this come from my current line of work in the Automotive Design industry. Many, many, many times over we can get overbooked on job quotes for designing tools, and frequently will export our work quote to outside companies. In the very early stages of this process, we simply went with the lowest bidders. However after time, we realized that the initial lowest cost did not mean the lowest final cost, as we ended up spending well over what others had quoted us for to fix the issues that were a result of cutting corners.