James Soto

2-10-2023

CS-320 Software Test Automation and QA

Project Two

The requirements were the main criteria and factor when it came to the code that I wrote. I wanted to ensure that every requirement, line by line, and the tests aligned correctly to ensure that I was not wasting time or energy on code that was not needed. Once I determined the requirements that were needed for each data structure, I used previous course assignments to build on each iteration of the classes, which helped me complete the assignment without having any problems. This course has taught me to use the information given in the rubric, create the requirements inside the software, and finally test the software using JUnit tests to meet the requirements I gathered from the rubric. Having this type of structure is in place is helpful on my path to becoming a software programmer while also setting a strong foundation in my academic career.

The JUnit test was effective due to every requirement being met with clean, consistent code that ensured that each tasks met the necessary criteria to pass the JUnit tests. Once every test was successful and every requirement was met, I knew that I used the correct JUnit tests. Each object contained requirements that needed to be explicitly tested by utilizing JUnit tests that met each requirement. One example is in the Contact Class, the contact ID string could not be longer than 10 characters, while nit being updated and not being null. When I ran the coverage tests on the different Services and on each of the Contact, Task and Appointment classes I had coverages between the 80%-95% range. The percentages being that high showed the effectiveness of the code that I wrote and of the JUnit tests.

I ensured that the code was technically sound by satisfying the requirements of the assignment. I wrote in a clean, error free style, while my code was syntactically accurate. In the code the if and while statements were written to satisfy the requirements for each task in the rubric. I also did a lot of testing throughout the process of writing the code to ensure that the code did not contain any bugs or errors that would require additional time to debug.

I ensured that the code was efficient by removing any unnecessary or redundant code. Once I finished writing the code I made small adjustments to see how the program interacted and determined if the small changes helped improve my code. I also reused code from previous modules to increase productivity. I used aspects of lean software development to ensure the integrity and consistency of the code while reducing time and waste. In the TaskTest.java test case, I created my starting variables which were the variables that I tested the code against. Every test that transpired in this test case section was to test specific cases in the code i.e.. getNameTest or setNameTest. By doing that each test case was only 3-4 lines of code. To continue the efficiency I used the same basic foundation from the TaskTest.java code and reused as much as I could to cover the aspects of TaskServiceTest.java while making changes on the necessary lines of code to meet the requirements within the rubric.

The software testing technique that I used for each assignment was White Box Testing, Unit testing, and functional testing. At the beginning of each assignment I used a lot of static testing to ensure the software was running free of bugs before I continued the development. Once the code was complete I performed dynamic testing, which ran the code to ensure the software was running as intended. With white box testing the person testing the code has an understanding of the internal structure and design of the software, and since I was the person who wrote the code I was white box testing my own software. Unit testing is testing every individual part of the software to ensure it works properly. Functional testing is testing that ensures the software matches the requirements of the assignment/project.

I did not use black box testing, since in black box testing the tester goes into the testing phase without knowledge of the internal structure or design of the code. Another testing technique that I did not use in my assignment was automated testing which would perform the testing without a user. Lastly, regression testing, which runs old test units against new software to ensure the software still performs as expected.

Let’s start off with black box testing, some projects might need an unbiased review of the software especially if they team uses someone who doesn’t have much knowledge of software programming, meaning they are testing the software without any structural knowledge and are able to give the QA team a different perspective on the overall software. White box testing lets the team test the software and the interactions between the test units, while not needing a complete UI to perform the tests. This helps on early in the development stage where the team is able to make the necessary changes to the code, which as we learned throughout this course, the earlier you find bugs or issues the cheaper and easier it is to fix. Static testing is a cost-effective way for the QA team to find bugs early in the development stage because in order to test during static testing the code does not need to be fully functional, again saving time and money. In regard to unit testing, it is essential that the unit tests are set up properly and are covering as many aspects of the software as possible to ensure that the development team does not suffer any setbacks the further the team gets into development. Functional testing ensures that the software meets or exceeds the requirements of the project. This is one of the last stages of testing, and in many instances uses black box testing to ensure unbiased testing.

After module 2 I wanted to increase my understanding of software testing and JUnit tests. My mindset was that the more I understood what to look for, the better my code and JUnit tests would come out. I took a course on Codecedamy with the intentions of doing just that. Throughout the milestones and Project one, my confidence and understanding grew with the more code and tests I wrote. I used the different complexities of each task, object and service requirement to improve the outcome of each test. For example, the first round of code I wrote, I completely changed once I realized that I had a better way of utilizing the JUnit and coverage tests. I wanted to have more coverage on every aspect of the code. I limited any bias I had when reviewing my code because I only utilized code that met the requirements and rubric. In a work setting I would have removed bias by having a coworker test the software instead of myself to ensure that bias was completely out of the equation. It is essential that the code we produce is free of bugs and errors, especially after reading the resources this week and learning about the major implications that have occurred when faulty or bad code was released to the public, where some institutions lost millions or billions of dollars, rockets or jets exploded, the death of people in extreme circumstances. In more realistic circumstances, software released with bugs or faults would causse crashes or performances issues, information could be leaked or stolen, applications would not properly work. In all those instances, the integrity of the company and the software team comes into question as to why they did not properly tests the software, thus creating more rework and money to fix the errors or bugs.