Andrew Wepplo

CS-320: Software Test Automation

Southern New Hampshire University

August 13, 2021

Project Two

My unit testing approach for each of the three features aligned with the software requirements because I started with creating the tests that the rubric mandated for all six of the standard code requirements. For example, all the databases had a requirement on character length, from 10 to 30 characters and that the field shall not be null, after implementing that into my code, I added the function to make almost all the objects in the contact service class be able to add or delete contacts or to be updatable for the four fields required in the rubric. When it came to the Junit tests in the additional files I ensured all of the fields for the set up were clear and organized for what was and wasn’t allowed in the inputs and made sure I kept an order to my tests so I could look at it in the form of a list from top to bottom for the features I was looking for if any errors were found in the running of the tests, luckily I only had a few mistakes in format such as typos that were easily fixed and all of my tests in each module finished without failures or errors.

To ensure my JUnit tests were technically sound and efficient, I followed the resources we were given on the proper set up and use for the Junit, I also followed some YouTube tutorials on how to complete the basics correctly, and I would say they are efficient because they complete the tasks required of them in the rubric without excess lines of code or unnecessary compiling such as in my delete test, or updating testing as shown below.

Graphical user interface, text, application

Description automatically generated

The software testing techniques I used in this project include performance testing with ensuring the code functions as intended, as in unit testing, integration testing to compile the four java files that communicate correctly for the compressed Zip file, and obviously the J-unit testing for each of the two java files for the past three weeks with the repeatable tests used to guarantee small things such as not exceeding a number of characters in a given string or description.

When it comes to a software testing technique that I did not use , officially I did not use integration testing and some other techniques I did not use include Ad hoc testing, agile testing, Black, White, and grey box testing, or even usability testing to see how the system works with its intended user base, but that testing process comes later when a project is almost ready to be delivered to a customer base. With these techniques, there are many practical uses such as in black box testing could be used in a security system to get a new bug fix out to a system without having to take it down for an extended period of time, or white box testing on an application to test a block of code as it is developed to make sure each function works as intended before moving to a new line or block of code so save time and money in the long run of a project.

The mind set I adopted for working on this project was still in the general realm of a student, meaning I did not have the feeling I probably would if I was doing this for work and my performance being seen by many others, but I did try to imagine I was a software tester and what would be expected of me in testing the code for Grand Strand Systems. In the idea of bias in review of code that I made, it’s hard to imagine what it would be like if I didn’t spend the time on the code, for example, I may have been more critical on certain facets of the JUnit tests, and I probably would have employed more testing techniques than I used in the end with the requirements of the code in this class. I don’t think that a tester should be responsible for testing their own code in a real-life scenario because they may overlook something because they assume it is fine because they were the one to write it and it may have had issues from the beginning that they overlooked from attention being directed elsewhere. I just think to not cut corners and to be professional in software engineering, there needs to be a system of checks and balances on code, testing, and implementation, to ensure there is a level of perfection in the code being written and not only being seen by one set of eyes, that can easily overlook an aspect of the code.

Lastly, when it comes to avoiding technical debt in the field, as we have learned, testing early and often is a great way to ensure that errors and bugs are caught early to avoid costly changes in the final steps of the project timeline just thinking it may have been easier or saved time in a certain phase but causes major problems, and ultimately the goal of getting the project completed correctly and on schedule is the only thing the consumer and in turn the provider of the software cares about being accurate in the end.