Devin Lemasters

2/16/2021

CS-320: Software Test Automation & QA

Southern New Hampshire University

**Summary and Reflections Report**

This report will summarize and reflect on the work I performed developing and testing Project One. I wanted to make sure that my approach to designing the code was properly aligned to the software requirements. I did this by looking at the requirements given to me and directly converting those requirements into the appropriate code. For example, I was able to take the requirements of the contact class for the contact service and align what was the requirements were to incorporate them into the code. Specifically, I was instructed to make sure that my contact class met the requirements of having an ID, first name, last name, phone number, and address. On line 5 of Contact.java, I declared those values as private string variables. I then created the getter and setter methods associated with those variables for modifying and retrieving the data. I also made sure the constructor incorporated the set methods to reduce code redundancy. I also made sure to incorporate the requirements of these private variables. For example, I was able to make the setId() method private so that it could not be modified outside of the constructor. I also made sure that null and lengthy inputs are invalid as per the requirements of the software. I handled the requirements for the other classes in a similar fashion.

When I was designing the JUnit tests, I wanted to cover testing as much of the code as possible. I am happy to announce that I was able to perform one-hundred percent coverage on all the classes under the main package. This means that all my classes have been fully covered. This ensures maximum quality assurance. I know that every line of code has been tested in its’ entirety and is working as expected. I ran tests to check for errors when there should be errors as well as checking for code to run without any errors.

I had a great experience with writing the JUnit tests. I was able to utilize different techniques to ensure my code was technically sound. I did this by optimizing my code where I could. My code includes documentation by providing comments with the code to guide with code navigation and understanding. I was also able to make my code efficient. This can be seen by referring to my previous example of including the set methods in with the constructor to reduce code redundancy. Another example of this can be shown in the Task class of the Task Service (starting at line 13) and the Appointment class of the Appointment Service (also at line 13) where I included these methods to fill the local variables of the class. An even better example of technically sound and efficient code is shown in the Task Service. On line 46 of the TaskService class I created a method for updating the task called updateTask(). This method performs different tasks depending on what information is passed to it. The class updates either the name or description of a task that is determined by the updateName() and updateDescription() methods. If updateName() is called, it runs the portion of code that updates the name. If updateDescription() is called, it updates the description instead. This reduced code redundancy.

I used a few different software testing techniques that made my tests successful. The main test method used was Assertions. Assertions allow me to assume that something should or should not happen. For example, assertThrows() is a method that expects there to be an error thrown. This lets me to easily test for invalid arguments and entries. I know that if I put in a null argument it will throw an error as null is not allowed for any of the class variables. The assertTrue() method assumes that a Boolean value is true. This means that I can check the variables I just set and do an comparison to make sure that it is set correctly.

For these milestones, I decided to stick with using assertions. The methods provided by this class was all I needed to use to prove that my code works. However, there are other strategies to testing that I did not use. Another technique that I could have used is called TestResult. TestResult is a useful utility that collects test results. This allows the collection of test results and distinguishes between failures and errors. The main difference here being that a failure is something that is a result of a failed assertion test, and an error is what happens when you get some type of exception, like an IllegalArgumentException.

These tests can be implemented in different software development projects and situations. In any type of situation, someone can run assertTrue() to check to make sure that something is true when it is expected to be. They can also use the assertThrows() method, like I did, to check for an expected error. These tests would probably work well in a database situation where storing data in a correct format is important or even to be used with storing data in certain types of accounts. This is also true with the TestResult example. For example, the TestResult could be used for an application that checks for certain errors and failures and reports them back to the user.

As for reflecting on my mindset during this process, I went into this project with an optimistic and open mind. After all, the project development from the start was my introduction to creating my own test cases. I was eager to learn how to effectively debug my code. I was cautious with my test cases as I wanted the tests to make sense compared to the actual code. I did this by implementing test that were necessary to prove that the code worked and made sure the order of the tests made sense. I ordered the tests in a way that would be replicated by a user to get the best results. It was important to appreciate the complexity and interrelationships of the code because there are various ways a user can input data that would have the potential of causing errors. For example, I had to check for each scenario that could cause an error, like those errors that show up when invalid or null data was set.

I put on a different persona when I was reviewing my code. This was to help prevent bias. I changed from the programmer developing the code and hoping it would work, to a tester throwing different scenarios at the code trying to get it to fail. I could imagine some concerns with testing my own code. Everybody is different. There may be other perspectives to approaching testing than one person, especially the developer, could come up with. For example, a program may call for a certain range of integers to be passed in and the developer may try to test invalid ranges of integer values. Someone else may try to convert and pass a floating point that may not be converted correctly. This would bring light to an error the developer was not expecting.

It is important to stay disciplined to the commitment to quality as a software engineering professional. It may be tempting to cut corners, but it is not worth it. What may seem like a time saving maneuver could lead to a bigger waste of time when it comes time for testing and debugging, especially if that corner cutting code occurs in the future. It could be something minor like spending hours pinpointing where the error originated, all the way to a catastrophic error that causes a substantial financial loss. Life-threatening situations or injury could even occur depending on what type of system this code is designed for. For an example on what could go wrong due to software errors, there was an incident back in 1998 with NASA’s Mars Climate Orbiter. “The error caused the orbiter to miss its intended orbit (87 to 93 miles or 140 to 50 kilometers) and to fall into the Martian atmosphere at approximately 35 miles (57 kilometers) in altitude and to disintegrate due to atmospheric stresses” (Siddiqi, 2019). The orbiter’s cost was $125 million, which is a huge waste of money (and time) that could have been benefitted elsewhere in NASA.

I plan to avoid technical debt by not cutting corners and write quality code. Doing this will help avoid errors later in the project that may cause more time and money to find later as opposed to doing it right the first time.

**References**

Siddiqi, A. A. (2019, July 25). In depth: Mars Climate Orbiter. Retrieved February 17, 2021, from https://solarsystem.nasa.gov/missions/mars-climate-orbiter/in-depth/#:~:text=What%20was%20Mars%20Climate%20Orbiter,translate%20English%20units%20to%20metric.