Jennifer Rebella

4/18/2025

CS-320

Prof. Omar Toledo Lopez

Project Two: Reflection

Summary:

The approach I took while creating my unit testing for the appointment, contact, and task services was to break down the software requirements and take each requirement and turn it into a multiple pass-fail test. For example, one of the requirements for the contact file was that the string field iD could not be null, longer than 10 characters and was unable to be updated, so in the contactTest I created tests to see if an error would be thrown as the code was programed to do if the iD was null and was over 10 characters. I then created a test to see if the iD would be successfully created if the requirements were met as well as a test to see if the contact would be created successfully. The quality of my JUnit tests was great. The coverage of the JUnit tests for all my Java files was either 80% or higher, which is the acceptable percentage for most tests. My experience with writing the JUnit tests was enjoyable. I found it an amazing tool to help continuous integration since an interface does not need to be fully developed and functioning before the code of the application can be tested. I did not struggle too much to understand the language and syntax used to perform the test. Any struggles I did have the provided resources or videos I found on YouTube explaining the concepts were enough for me to understand them. To ensure that my code was technically sound and efficient I used the industry’s best practices and added comments to my code. Comments help explain what I want the code to do so that if another developer works on the code, they will understand what the code is supposed to do. Another way I made sure my code was technically sound and efficient was the use of ifelse and for loops. Using this help the code run with less lines and prevents human errors that can result with very long code files.

Reflection:

The software testing techniques that I employed for this project were static and dynamic testing. Static testing is looking for human errors like spelling and syntax errors and not logic errors from the software. Static testing is done without running the code. Dynamic testing is testing done while the code is running. This type of testing looks for logical errors made in the code and any unexpected outcomes that the software might produce. Specifically, I used JUnit testing to test that the software runs as required. One software testing technique I did not use was security testing. Security testing is when you view the code for any security vulnerabilities. Examples of security testing include dependency reports and Maven tests. I did not use this type of test since the requirements did not specify that the code needed security tests, the only security requirements is that the id is unchangeable.

Mindset:

While working on this project I adopted a calm mindset. Though the software had many requirements since the work was broken week by week it was much easier to digest, and work threw. Remaining calm and collected made it easier when I ran into errors and issues since I did not get overwhelmed it was easier to find the issues and think of the solution. It is important to appreciate the complexity and interrelationships of the code since understanding this will make debugging and testing a less daunting task. Understanding that the code is complex and is related to many different parts of the software will make you prepared for the challenges that come with testing. I tried my best to limit my bias on my review of my code by following feedback provided on my software and taking them into account. Testing your own code can lead to oversight of problems in the software. Since you know how you want the software to function and what processes you used to achieve this, it can lead you to think your code is clear and concise but others without this knowledge may struggle to get the software to run properly especially if the user has no background in code. It is critically important to keep yourself disciplined in your commitment to creating quality software as a software engineering professional. Cutting corners while writing and testing codes can result in many errors which lead to lost time and money. To avoid technical debt, I would use continuous testing. Continuous testing is a great way to make sure that the code is functioning as expected threw out the whole development process since the code is tested each time a new part is added. This allows us to know that we are meeting requirements and deadlines.