Summary & Reflections Report

Jacob Bennett

SNHU

CS 320

Summary

Unit testing was conducted for each class within each of the three features. My approach to testing was directly aligned with the software requirement. I started by designing code and writing code in their respective classes. I then developed test cases for each method in the classes. The goal was to get above 100% code coverage. In most of the classes, I hade around a minimum of 80% code coverage. The below screenshot shows us a test to delete a contact from the contact functionality project. 

The screenshot shows us that we were able to add a contact and delete that same contact. If it didn’t delete the contact, it would fail the test. One edge case I missed is if there were unhandled exceptions in the creation of the contact. The quality of the Junit tests are an important factor. We have a high coverage percentage. We know that the Junit tests are effective because they cover most of the edge cases. A computer screen shot of text

Description automatically generated

In this screenshot it shows us that we have three test cases all to test the address requirements. This not only ensures we have a high coverage percentage but also shows that the tests are effective.

I ensured my code was technically sound by following best practices for coding.

A screen shot of a computer program

Description automatically generated

I have methods for getters and setters in every class I design. This is an example of technical sound code. By not directly modifying global variables from class to class, we ensure we have known behavior. Ensuring code efficiency is difficult at times. A screen shot of a computer code

Description automatically generated

The screenshot shows us that we are accounting for all edge cases when entering a phone number (xxxxxxxxx, xxx xxx xxxx, and (xxx) xxx-xxxx). This cuts down on the amount of code required to check a phone number and therefore is more efficient.

Reflection

In this project manual testing, functional testing, and unit testing were all heavily used. Manual testing is manually inspecting and testing the software. Functional testing tests the functional requirements of the program are met. This type of testing was heavily used as it was the basis of test in my strategy. The last kind of testing, unit testing, tests individual components of the software. In our case it was test each method to ensure that they function as intended.

Other testing types not used in this project include integration testing, system testing, security testing. Integration testing is the art of testing ow each component interacts with each other to form a complete system. System testing tests the completed system to ensure that it meets requirements. Security testing tests the system of software for known vulnerabilities, finds defects and can include a mitigation plan.

The practical uses for manual testing, functional testing, unit testing, integration testing, system testing, and security testing are immense. Manual testing has a low bar of entry and therefore is a great tool to use for “at-a-glance” sessions. Functional testing and unit testing usually go hand in hand as they test a specific piece of the software against requirements. Integration testing is a very practical test utilized to make sure the “pieces” of the software work together as intended. System testing and security testing I have some real world experience. I test space command and control systems at work. A large part of the operational test is making sure the whole software system works together. Security testing is paramount to ensuring our systems are safe. This type of testing is sometimes overlooked as it can become very expensive and very difficult to fix exposed vulnerabilities. The mindset I carried throughout this project was one of “detail-oriented”. As a software tester, we must employ some level of caution. In this role it is easier to spot every little problem, but we must step back and look at the whole picture to ensure that we are meeting the intent of the system. Its important to look at the complexity and the interrelationships of the code that we are testing. An example of this is in our last project where we put everything, we’ve been working on for weeks together. The relationship between the code would have been immense and would have required a complex driver program.

Limiting bias can be difficult, more so when it comes to your own code. I keep an open mind and dissect differing ways to solve a problem to better my own code or processes. Discipline is an important skill to hone in a professional and personal manner. Ones own commitment to quality as a software engineering professional is paramount to your own image and how you’re viewed professionally. Cutting corners when writing and especially testing code can be very tempting, it is important to step back and think of the ramifications of one’s actions. I personally plan to avoid technical debt by utilizing project management techniques. Even now I employ miniature scrum meetings (with myself as the only member). This helps my prioritization of tasks that need to be done by a certain deadline.