**Project Two**

Grace Fletcher

Southern New Hampshire University

CS 320: Software Test, Automation

Prof. Federico Bermudez

April 15th, 2025

**Summary**

For Project One, there were three features that needed to be tested: Contact, Appointment, and Task. To meet the software requirements, I create four different classes for each feature. For example, for the Contact feature, I created a Contact class to hold contact information, a ContactService class to edit contact information, and test classes for both the Contact class and ContactService class. The ContactTest class made sure that all variables aligned with their respective requirements. For example, a contact’s first name could not have more than 10 characters, so I added in a test to ensure this requirement was met. The ContactServiceTest class tested to ensure that all methods worked correctly. The logic used for the Contact feature’s classes and test classes was applied to both the Appointment feature and the Task feature.

I used JUnit testing for class testing and test class testing to make sure that input and methods were handled and running correctly. As I mentioned before, the tests I ran on the base class (i.e. ContactTest) tested to make sure that the variables/input met software requirements while the tests I ran on the service class (i.e. ContactServiceTest) tested to make sure that methods dealing with variable manipulation ran correctly. For every test in this project, an error is thrown if the test is failed. For example, when testing the “addContact” method in the ContactService class, if the contact being added is not found after the method is called on, the program throws the message “Contact was not added correctly.” to the console. When it comes to test coverage, I aimed for 80% test coverage in my project files.

When writing the JUnit tests, I made sure that my code was technically sound by leaving in-line comments. This improved the readability of my code, and the comments acted as a sort of label for each test. I also utilized the JUnit DisplayName import for each test. For example, for the JUnit test of the “addContact” method in the ContactServiceTest class, I added an in-line comment saying “Testing addContact” and made the DisplayName annotation say “Test to make sure adding a contact works.” This made it easy for me to scan through my code to make sure I was covering everything that needed to be tested. Another way I made sure that my code was technically sound was by following coding best practices when it came to formatting, spacing, and commenting etiquette.

Leaving in-line comments and following coding best practices also made my code more efficient. I also made sure to avoid redundancy in my code and tried to use methods and variable declarations that were as compressed as they could possibly be. One example of this would be my use of the JUnit assertEquals method import. This method allows for an operation that would typically take 3 or 4 lines of code to only take 1 line of code. For example, when testing the “updateAddress” method in the ContactServiceTest class, I used the assertEquals method in the following way: *assertEquals(“123 Pleasant St.”, service.getContact(“15”).getAddress(), “Address was not updated.”);*. This method compared “123 Pleasant St.” with the address located at index 15 of the contact list array and, if they didn’t match, sent out the message “Address was not updated.” to the console, all while being contained on one line of code.

**Reflection**

I employed two types of testing for this project: static testing and JUnit testing. Static testing involved me scanning the code visually line-by-line to make sure everything looked correct and free of errors. I implemented JUnit testing by creating separate test files for each class (i.e. Contact, ContactTest, ContactService, ContactServiceTest). Within these test classes, I created tests that would test various inputs to make sure I was getting the desired output. This is also where I ensured that different variables and elements of my code met software requirements.

One software testing technique that I did not use would be performance testing. According to a software testing guide on BrowserStack, performance testing is “a type of software testing that assess the performance and response time of a software application under different workloads.” (BrowserStack, 2024) An example of implementing this type of testing would be to run the “addContact” method and measure the resulting load time of this method execution. Another technique that I didn’t use is security testing. According to the same guide, “Security testing is a type of software testing that assess the security of a software application.” (BrowserStack, 2024) An example of implementing security testing would be the use of penetration testing, which involves simulating an attack on the software to exploit potential vulnerabilities. (BrowserStack, 2024)

I used a lot of caution when acting as a software tester for this project. The reason for this caution was because I wanted to avoid as many errors as possible. If I became lenient or negligent at any point in coding or testing, I would likely run into costly errors that would impede on project progress. It was important that I knew how each piece of code interacted with one another to test the code effectively. For example, when running JUnit tests on “updateAddress” in the ContactServiceTest class, I had to know what other methods from the Contact feature were being used in order to successfully test the “updateAddress” method.

One way that I tried to limit bias in my review of the code is by treating the code as if it were presented to me for the first time and that it wasn’t my code when running tests. I feel that bias could be a concern when running tests on your own code, as you don’t want to think of yourself as someone who messes up or causes errors.

It is important to be disciplined in commitment to quality as a software engineering professional for multiple reasons. One reason being that if a software engineer becomes negligent or starts cutting corners in their work, they not only impede on a project’s progress, but they can also cost their company a lot of money due to things like errors and redundancies in their code. I plan on avoiding technical debt as a software engineer by using coding best practices and properly implementing software testing techniques. One example of a software testing technique that I plan on continuing to use would be JUnit testing. I’ve found that this technique/implementation works the best and covers the most ground when it comes to software testing and quality control.

**References**

BrowserStack. (2024, August 22). *Types of Testing: Different Types of Software Testing in Detail*. BrowserStack. https://www.browserstack.com/guide/types-of-testing