**Project Two: Summary and Reflections Report**

Blake Kemp

Southern New Hampshire University

CS-320 Software Test Automation

Professor Bolton

April 16, 2023

Over the last few weeks, our focus has been on studying and implementing unit testing techniques for modules developed for mobile applications. Although the assignments were comparable in structure, we encountered varying requirements for each module. To accomplish this task, we employed JUnit 5 library, which proved to be an invaluable tool in simplifying the creation and automation of our testing process. We learned that JUnit 5 is a powerful open-source testing framework for Java that has many built-in features, such as parameterized tests, test instance lifecycle methods, and assertThrows() methods, which make writing tests quicker and easier. By utilizing the JUnit 5 framework, we could easily identify defects early on in the software development process, which helped to reduce the overall cost of development and improved the reliability and quality of the mobile application.

It is crucial to keep the software requirements at the forefront of your mind when testing various modules to ensure that they meet the necessary standards. Running tests against the software is the best way to verify that the requirements are fulfilled. Throughout the course of each assignment, I made certain to test the specific requirements associated with each module. The initial module I worked on was the Contacts module. This module involved multiple variables to store data, but the data had certain limitations. For instance, the contact’s last name was required to not be *null* and have less than 10 characters. Therefore, I created a validation check for the contact’s last name and then carried out tests to ensure that the check was effective.

setLastName function: A picture containing text, screenshot, font, line

Description automatically generated

The associated Junit 5 Test for the last name function:   
A picture containing text, font, screenshot

Description automatically generated

The setLastName method sets the last name for a Contact object. It first checks whether the provided lastName string is null or has a length greater than 10. If either of these conditions is true, it throws an IllegalArgumentException with the message "Invalid last name". Otherwise, it sets the lastName instance variable to the provided value. The testSetLastName JUnit 5 test checks that this method behaves correctly under different scenarios. It tests that setting a valid last name updates the Contact object's lastName field as expected. It also tests that attempting to set a null or an invalid length last name results in the expected IllegalArgumentException being thrown.

I can confirm that the task and appointment modules were like the contacts module, but with different requirements. The task module necessitated an object with a unique ID, name, and string that each had constraints. I tested these constraints to guarantee that they failed when the limits were exceeded. Similarly, the appointment module demanded an object with a unique ID, a description, and a Date object with additional testing. The appointment date couldn't be set before the current time, and it couldn't be empty. I examined each constraint on these variables until they failed. Moreover, using JUnit 5 tests, I verified the service classes for each module, which used a HashMap to organize the data.

Ensuring that I covered 100% was paramount to knowing that all avenues were covered.   
To ensure that the code is technically sound, JUnit 5 tests were used to test various functionalities of the TaskService class. For instance, the testAddTask() test ensured that a task is added to the task list and can be retrieved by its ID. This was done by creating a new Task object, adding it to the task list using the addTask() method, and then calling the getTaskId() method to retrieve the task object with the specified ID. The assertEquals() method was used to assert that the task retrieved is the same as the task that was added. Similarly, the testAddTaskWithDuplicateId() test ensured that the code throws an IllegalArgumentException when a task with a duplicate ID is added to the list. The assertThrows() method was used to ensure that the exception is thrown when this occurs.

To ensure that the code is efficient, the tests were designed to cover various scenarios and edge cases. For example, the testAddMultipleTasks() test ensured that multiple tasks can be added to the task list and retrieved by their IDs. This test covered a scenario where the task list contains more than one task. The testUpdateTask() test ensured that a task's name and description can be updated using the updateName() and updateDescription() methods. It also tested that these methods can handle null inputs. Finally, the testDeleteTask() test ensured that a task can be deleted from the task list using the deleteTask() method. This test covered a scenario where a task is removed from the task list. All these tests were designed to ensure that the code is efficient and can handle different scenarios and edge cases.

testAddMultipleTasks test:   
A picture containing text, screenshot, font

Description automatically generated

To ensure the quality of the code, it was important to have a good understanding of different software testing techniques. The main techniques used in this semester's testing were unit, static, and dynamic testing. Unit testing focuses on testing individual components of the application, ensuring that errors are not leaking from one unit to another. Static testing involves manually or automatically reviewing the code without executing any code. This type of testing was performed before running the applications to identify any syntactical, technical, or efficiency errors. Dynamic testing involves executing the code and verifying that the output is as expected. Most of the testing done in this project utilized dynamic and unit testing techniques.

During my testing process, I utilized a few techniques such as unit, static, and dynamic testing. However, there are other testing techniques that I did not use this semester, including integration, system, and acceptance testing. Integration testing involves testing the application by combining all the modules or units and verifying that they work as intended. System testing, on the other hand, involves objectively testing the entire system to ensure that it meets functional and non-functional requirements. Due to the need for an unbiased perspective during system testing, sometimes independent teams are brought in. Lastly, acceptance testing is performed to ensure that the software meets the main objectives or requirements. The most common type of acceptance testing is user acceptance testing, where users themselves test the software to ensure that their needs are met.

As a developer, it can be easy to fall into the trap of assuming that you know what you're doing and may not need to create simple tests for your code. However, it's essential to create individual tests for each attribute and object and avoid bias. This is why documented requirements need to be given to a development team, and independent testing teams may be necessary. I understand that discipline is crucial when showing my work. It's essential to avoid taking shortcuts that may lead to code that is challenging to read, fix, and add onto later. Therefore, I'll make sure to create high-quality code from the start by employing efficient coding techniques, adding comments to my code, and assessing vulnerabilities via dependency checks to increase security when necessary. I'm grateful for what I learned during this course, and I'm confident that it will make me a better developer.