Tanner Humphreys

CS320 7-2 Project Reflection

Over the past month I have been given the task of developing a mobile application and creating unit test for several aspects surrounding the project. This project was broken into three different sections with each section focusing on one part of the overall application. These sections were named Contact Service, Task Service, and Appointment Service. Each section had various requirements and each requirement needed to be verified using JUnit tests.

The first part of the application was the creation of the contact class. The class created an object from several string variables including a unique ID, first and last name, a phone number, and an address. Each of these variables had stipulations that needed to be tested to ensure quality. For example, the first name could not be longer then ten characters, and the ID associated with each contact needed to be unique upon creation. Accompanying the contact class was the contact service class. This class was used to manipulate the objects within the contact class. The purpose of this class was to update, add, or delete contacts based on unique ID. The manipulation of these objects also needed test to ensure the best quality.

Much like the first part of this project, the task service portion also created objects based on client requirements. These requirements included the use of unique IDs, a task name, and a task description. Just as in the contact class, the ID needed to be unique, the name could not be longer than twenty characters, and the description could not be longer than fifty characters. To manipulate these objects, the task service class was made. This class could add, remove, and update tasks based on the ID.

The final section of this project was the appointment service class. This class was created using the same techniques as the first two parts. This class used a date for creating the appointment. A unit test was created to check these dates because a requirement for this class was that the appointment date could not be from the past.

A large part of software development that is often not thought about is the amount of testing required to guarantee quality of the final product. With each of the classes created, a JUnit test needed to be created to test each part of the class. As part of the project, I was given the requirement to ensure the test coverage for the files was at 80% or higher. I was able to complete this requirement because I wrote a test for each variable. For example, the first name of a contact could not exceed ten characters in length, and this variable could not be null. I created a test to first check if the first name was null, and another test to check if the input was greater than ten characters. If the name was null, or if the name was longer than the length requirement, an error was given. This type of testing was used for each variable for each class. An example of a passed test is given below.

@Test

**void** updateFirstNameTest() {

ContactService service = **new** ContactService();

service.updateFirstName(service.getContactList().get(0), firstNameTest);

*assertEquals*(firstNameTest,service.getContactList().get(0).getFirstName());

}

This test will update the first name by first creating a new ContactService object named service, and then call the updateFirstName method. This method takes in a contact within a list and new updated name. The test then asserts that the first name of the given contact is equal to the updated name. This method of testing continued throughout the project for each variable within each class.

While there are many ways to test code, I was tasked with using JUnit test throughout this project. There was no UI creation in this project, so the use of JUnit testing was an efficient way of testing. In past projects I have used terminal outputs to follow variable manipulation during run time, or even pop-up windows assuring that code has worked properly. In a past class at Idaho State University, I created the game of Monopoly that up to eight people could play at once. This project was heavily focused on graphical output as well as text output. I did not use unit testing for this project, but instead I used a large text box that gave an output of every action that happened within the game. In the beginning of the project, it was used for testing and quality assurance, but was later used in the final version of the application to help players better follow the game. It showed what value of the dice was rolled, where the player advanced to, and to let players know what options were available during their turn.

The project for this class was rather cut and dry with each class and test being almost identical. My first approach to this project was simply create the required classes and then run basic tests to see that everything was working properly. I started by writing what I consider bad code. In the first part of the project, I used hard values in serval places. While this method of coding can work, I realized that it was not the best way. For example, several variables could not be more than ‘X’ number of characters. I wrote code such as: if(firstName > 10). While there is no errors with this code, if future developments said first names could 15 characters in length, many changes would need to be made to the code. Instead of hard coding values, I created final values that can be changed with ease if the value requirements change. For example: **private** **static** **final** **byte** ***CONTACT\_FIRST\_NAME\_LENGTH*** = 10;

Testing is a skill that every member of a development team should have. Throughout this course I have gained a better understanding of what testing is, and how to apply a variety of test within my code. At some point I know that I will not be the only person working with code for a project, and I have learned how to develop code that is easy to understand and test.