Corey Sampson

CS 320

April 19th, 2024

Dr. Kraya

7-2 Project Two

**SUMMARY**

For the development of the mobile application there were three features the customer required to have implemented. Those features are the Contact Service, Task Service, and Appointment Service. Each of these features had specific requirements provided by the customer. To ensure the requirements for each feature were met, writing dependable code for testing was critical. For example, one of the requirements within the Contact Service feature was that the Contact Class has required first and last name fields that can not exceed 10 characters in length or be empty (null). In my code, I met these requirements by creating a contact object with the field’s first name and last name. In this object creation I provided if statements against the requirements, for example:

if (firstName == null || firstName.length() > 10) {

throw new IllegalArgumentException("First Name cannot be more than 10 characters or empty.");

}

Basically, this bit of code tells us that if the first name is empty or greater than 10 characters, an exception is thrown to prevent that from being valid. This object creation is then followed by setter and getter methods to set the first and last name, and then retrieve them if they need updated.

When getting to the Junit testing part of my code, I made sure to implement coverage across multiple boundaries (value boundary testing) by implementing faux contact creations using the max amount of allowed characters, and characters in between null and the max amount. Based on my coverage percentages, my JUnit tests were efficient and effective as every category in my testing for all three features, was at 100% in my main package (full test coverage can be viewed below in the appendix). As mentioned previously, to ensure my code was sound within my JUnit testing, I made sure to cover every bit of code in the java files in my main package of the project. Those files included the Contact Class and Contact Service Class. Technically sound code was implemented using strings and arrays for obtaining and storing the Contact information. This was done by using the following code:

public static ArrayList<Contact> *contactList* = new ArrayList<Contact>(0);

This bit of code imports an array for storing data. In this case, the array was used to store data for a contact object (such as ID, firstname, lastName, etc.). One way I made sure efficiency was present in my code was by declaring each variable before using it. By disclosing variables for the Contact Class before comparing them to requirements, I ensured that each variable is efficiently written. This is example can be seen in the code below:

private String id;

private String firstName;

private String lastName;

private String phone;

private String address;

// Contact object of Contact Class

public Contact(String id, String firstName, String lastName, String phone, String address) {

**REFLECTION**

The most used software testing technique used in my code was White-Box Testing. This technique involves testing the internal logic and structure of the software. The JUnit tests were designed based on the structure of the code. For example, the Contact Class has a counterpart class called Contact Test Class that provides JUnit tests based on the code in the Contact Class. Implementing JUnit tests is another form of software testing I used, known as Unit Testing. This methodology involves testing individual units in isolation. One example from my code is specific JUnit tests for each of my setter methods. Lastly, another technique used throughout my code is Boundary Value Testing. This technique ensures that when specific requirements are applied, such as character limits, there is testing on more than just the maximum. In my Contact Test Class, I had two separate Contact creation tests implemented. One used the maximum values for character limitations, while the other used character values less than the maximum to ensure coverage between the maximum limit and the minimum limit.

One software testing technique I did not utilize in my work was Black-Box Testing. This technique involves testing without knowing the internal structure of the software. This was not an option as I was not provided with a code base and the application was built from scratch. However, if someone took the code base I created, they could use the Black-Box method of testing to test my code further. Another technique that went unused in my application was Performance Testing. Although important with application building, Performance Testing was not an option since there was no interface connected to my mobile application. Performance testing evaluates responsiveness and stability under various conditions of workload. Each of the techniques I mentioned plays a crucial role in various projects for ensuring early detection of bugs and errors throughout the software development lifecycle.

My mindset changed multiple times between the first Milestone and before submitting the final project. Each step up to finishing the project I employed more and more caution. I gained an understanding of how the interrelationship between my main classes and my test classes affects the overall security and efficiency of my code. Understanding this relationship helped me be more cautious in my coding as well by ensuring each test covered each of the working components of my code. This allowed me to also ensure that my code coverage in my JUnit tests were no less than 100%. To eliminate bias when reviewing my code, I put myself in the shoes of the customer based on the requirements of each feature. While this does not eliminate bias, it helps. However, I feel it is important for developers to not test their own code due to the varying level of bias and how that can affect the overall quality of the software. Lastly, being disciplined is one of the more important factors to consider as a software engineer. Cutting corners while coding can decrease the overall quality of the software and potentially cause bugs or errors to be omitted or missed. For me, to avoid these issues, testing early and often can help reduce bugs and errors, as well as bias, and ensure a better, higher quality product to present to a stakeholder/customer.

**APPENDIX**

*Appointment Service, Contact Service, and Task Service Test Coverage*  
(Search function is not 100% due to branch going unused)

**A screenshot of a computer program

Description automatically generated**