7-2 Project Two

Ahmed Sharairi

Southern New Hampshire University

CS-320-R4836

Prof. Karl Lewis

April 17th, 2024

I made sure my test cases were directly lined up with the software requirements. Each case tested the requirement that was given using a function that would detect if the requirement was not met. For example, a requirement in the task service class stated that the name could not exceed twenty characters or be null. In my code, I made a function that checked if the name was more than twenty characters or null. Then, I made a test case that purposely made the name too long and null to see if the system would pick it up.

I know that my JUnit tests were effective for both services because I specifically wrote test cases to check certain aspects of the code. These tests came back successfully which means each case correctly executed and checked the code. Furthermore, the percentage of coverage was above 80% which meant that the test cases checked a good amount of the code.

I ensured that my code was technically sound by making sure to follow industry standard practices when writing my code. This involved making sure to privatize some variables when needed, and making sure that I wrote separate code for the getter, setter, and constructor functions. Although I could have made shortcuts, I chose not to in order to keep the program's integrity. An example of this is from my contact class, in which I made a contact variable, made a setter function, implemented it into the constructor, and made a getter function.

```
private String contact_ID;
public void setContact_ID(String contact_ID) {
 if (contact_ID == null || contact_ID.length() > 10) {
 throw new IllegalArgumentException("Invalid ID");
 }
 this.contact_ID = contact_ID;
}

setContact_ID(contact_ID);
public String getContact_ID() {
return contact_ID;
}
```

To ensure that code was efficient, I followed the same practices as I did for making sure that my code was technically sound. I made sure to write code in a way that followed industry standards while also being organized. On top of this, I made sure to cut out any unnecessary code when I could. An example of this is in my ContactService class, I simply define an array and add a contact to it.

```
// Creates an array for the contacts
private ArrayList<Contact> contactList = new ArrayList<>();

//Adds contact
public void addContact(Contact contact) {
contactList.add(contact);
}
```

The software testing techniques I used for each component in the project were similar because the code objective was the same. For example, in each class, there were specific variables that had to be checked. The first check was to see if the variable was too long or short, and the second check was to make sure that it was not null. The exception was in the Appointment classes where the date had to be checked to ensure it was not in the past. Beyond the variables, the service classes had to be checked to make sure that the code could add, delete, and update the contacts, tasks, and appointments. JUnit tests made checking the code very easy and efficient. Furthermore, I manually inspected the code to check for any human errors with the syntax.

A testing technique that I did not conduct was with the security aspect of the code. I did not employ any methods to check if the security of the data being interacted with in the application was secure. For example, I could have checked if the libraries I used are up to date, or if the variable inputs are validated to avoid cross scripting. Luckily, the assets I used were from Java's default libraries, so there should not be much of a concern there. However, security testing is still crucial if the application is going to be used commercially.

The techniques I utilized can be used for multiple projects and situations beyond the ones here. For example, JUnit testing is a very common type of testing that makes checking segments of code simple. In addition to that, checking code manually is a commonly used technique as well.

As a software engineer, I made sure to be cautious as I was writing and reviewing the code because ultimately, this product is for a client who will be paying. Making sure that the code flowed and related to each other was important in the testing phase especially. A lot of the code that was written was like each other, but their purposes were different. For example, the contact, task, and appointment classes all had the same functions and service classes, however their purposes differed. The contact class held individual information, the task class held information about certain objectives, and the appointment class kept information about appointments. All these classes work together to form the final product.

In trying to limit bias in the code review, I reviewed all possible instances with my JUnit testing. Bias is a concern in the industry because developers assume their code has little to no errors, and this mindset can be very dangerous. For example, in my code, I made sure to cover all possible errors, but if I didn't review them properly, it would be an issue because I could have missed something. Although I was not able to use this strategy during the development, peer testing is a very helpful form of testing. It allows a different set of eyes to review your work and catch any potential errors.

It is important to be disciplined in quality assurance of software development because the client may rely heavily on the software to run their organization. On top of that, yours or your company's reputation is at stake if poor quality code is consistently sent out. In addition to that, technical debt will occur if testers are constantly correcting easy mistakes in the software,

wasting time that could be allocated elsewhere. All in all, poor quality code affects both the client and the software development team.

In conclusion, by writing test cases, employing industry standard testing techniques, and by making sure that software functionality is prioritized, software can be developed for the client that meets their requirements.