CS 320

KaLee Li

Professor Waithe

April 13, 2023

7-2 Project Two

1. **Summary**
2. **Testing Approach**
   * 1. **Software requirement alignment**

The testing approach I used to code contact, task, and appointment service application aligned well with the software requirements. I performed an individual test case to test each condition to ensure the code was executed correctly. As you can see in the example chart below, there are five requirements and two requests in each requirement in the contact class, and I used 10 individual test cases to ensure my testing approach aligned with the software requirements.

|  |  |
| --- | --- |
| Contact Class requirements | Contact Test |
| 1. contactId CANNOT BE NULL, no longer than 10 characters, and not be update-able 2. firstName CANNOT BE NULL and no longer than 10 characters 3. lastName CANNOT BE NULL and no longer than 10 characters 4. phoneNum CANNOT BE NULL and must be exactly 10 digits 5. address CANNOT BE NULL and no longer than 30 characters | 1. testContactIdTooLong() 2. testContactIdIsNull() 3. testFirstNameTooLong() 4. testFirstNameIsNull() 5. testLastNameTooLong() 6. testLastNameIsNull() 7. testPhoneNumTooLong() 8. testPhoneNumIsNull() 9. testAddressTooLong() 10. testAddressIsNull() |

* + 1. **Overall Quality**

The overall quality of my Junit testing was quite effective based on the coverage percentage of 80% – 100%, which indicates that most of the written codes were executed. Screenshots of the percentage are provided below

|  |  |
| --- | --- |
| **Contact Service Application and Test Coverage** | |
|  | **Graphical user interface, text, application  Description automatically generated** |
|  |  |
| **Task Service Application and Test Coverage** | |
|  |  |
|  |  |
| **Appointment Service Application and Test Coverage** | |
|  |  |
| Text  Description automatically generated | Text  Description automatically generated |

1. **JUnit Tests**
   * 1. **Technical and Efficiency**

To ensure my codes were technically sound and efficient, I ran the code with Junit testing as I wrote the code. As a programmer, we should not assume our code is functional even without syntax errors. We should start testing early, use individual test cases to verify the code is executing as expected, and test all possible conditions to ensure the system behaves according to the requirements. For example, the user can update the task name using their task ID in the update task name method. So, I created 2 test cases for my method; First one was to test to ensure that the user could update their task name as per the ID. The other test case was to ensure the illegal argument was thrown when the ID was not found. The codes and test cases screenshot is below

Graphical user interface, text

Description automatically generated

Text, letter

Description automatically generated

1. **Reflection**
2. **Software testing techniques**
   * 1. **Software testing techniques**

For the software testing techniques, I employed black-box and white-box testing, integration, and unit testing for each milestone. Details for each milestone are listed below:

|  |  |
| --- | --- |
| Contact.Java | |
| Text  Description automatically generated | Text  Description automatically generated |
| One of the requirements in Contact.Java is that the contact ID can not but null and can not be longer than 10 characters. Therefore, I wrote two test cases to determine if the contact ID is too long and if the contact is null. The requirements were fulfilled if they passed after I ran the Junit test. | |
| TaskService.Java | |
| Text  Description automatically generated | Text, letter  Description automatically generated |
| One of the methods in the TaskService, update the task name. The task name is updatable as per the unique ID. I wrote two test cases to test the update task method to ensure it worked. The first test case was to test the update task name as per ID. First, I added a task to the object and then tested the update task name method with the same ID with a new task name to see if the new task name was updated. The second test case was to test if there was no matching ID and if the illegal argument would throw. Both test cases passed Junit testing and were executed. | |
| AppointmentSerivce.Java | |
| Text, application  Description automatically generated with medium confidence | Text, letter  Description automatically generated |
| A method in AppointmentService.Java can only add an appointment with a unique. If the ID was there, then we cannot add a new appointment. If it is not there, then add a new appointment. I wrote two methods to fulfill this requirement. One method is to ensure the ID is unique, and the other method is to run the unique ID method before adding an appointment. I have only two test cases to ensure these methods are integrated and functional. The first test case is to test adding an appointment with a matching Id, and the second is to test adding an appointment with a unique ID. Both test cases passed and executed. | |

* + 1. **Software testing techniques that you did not use**

I did not use non-functional testing techniques for the milestones. Non-function testing is to test performance, security, and reliability. We did not obtain non-functional requirements, nor did we test to see how the application performs under certain circumstances or the behavior using different operating system environments. All the milestones' applications use an in-memory data structure to support storing data. Thus, the elements that can be held in the array list to maintain performance are limited.

* + 1. **Practical Uses and Implications**
* Black box testing – I first review the requirement and specifications in the milestone document. I used different inputs to check if the output fulfilled the requirements.
* White box testing – I wrote individual test cases to test each code's path.
* Unit testing – I took a unit of the class and wrote individual objects in the test case to test the method individually.
* Integrating testing – wrote a few methods and tested them in a group.

1. **Mindset**
   * 1. **Caution**

This is the first java project I built from scratch, and I had to act as a developer and the tester, so I was extra vigilant. I adopted a process mindset before working on this project. I reviewed all the guideline documents, the milestones, the journals, and Project One and Two's documents to learn all the requirements, understand the complexity, and make connections between milestones and project one, the journals, and project two. Then, I estimated the efforts by rating the complexity of the project requirements and expected to visit the tutor. Also, I used the guidelines documents as my requirement checklist and ran the code frequently while writing them to ensure the code was executed correctly.

* + 1. **Bias**

I can imagine conscious and unconscious biased concerns may occur when a software developer has to test their code. Depending on the type of systems the developers are creating, personal and informational biases may embed into the code without realizing it; These biases can significantly impact society. Therefore, it is vital to have bias control to reduce bias as much as possible. For example, get another set of eyes to review your codes or get user comments and fix them accordingly.

* + 1. **Discipline**

In my commitment to providing quality as a software engineering professional, I tried to limit bias in reviewing my code by getting feedback from the professor and the tutors for this project. Also, if my code didn't pass the Junit testing, I would trace the Junit failure, figure out the reason, and repair the code to avoid bias. It could be frustrating and time-consuming when there is a problem with the codes and the need to do tracing and repair. However, it is important not to cut corners while writing and testing the code; start testing early to avoid spending more time tracing and potentially having to rewrite the code entirely. Fix the defects/errors before it gets implanted into the system. For example, I was having trouble running one of my test cases while developing the Task service in module 4. I took the extra time to email and visit the tutor for guidance. Fortunately, the error was an early finding; thus, it was an easy fix.