Testing plays a major role in the software development lifecycle. The goal is to test as much as possible to make sure the code is working properly with the best performance, minimal errors, and less vulnerabilities. My approach to testing the software requirements was to use the JUnit tests to test the given requirements for the project. Specifically, in ContactTest.java, I was testing the lengths of the contact ID, first name, last name, phone number and address. Then again to test if a spot was null. In ContactServiceTest.java adding, deleting, and updating a contact was tested. In Appointment.java I tested the length of the ID and description, tested the date to ensure it was not a date in the past, and tested if a spot was null. In AppointmentService.java I tested the add and delete functionality. In Task.java I tested the lengths of the task ID, name, and description lengths were tested and to check if a spot was null. Lastly, in TaskService.java the add, delete, and update functionality was tested. The JUnit tests were able to cover 90% of the written code which shows how they are effective when testing the software requirements.

I made sure my code was technically sound by using easy to understand variables when writing this code. I also used four packages. The package names are appointment, contact, task, and test. The packages are made so the different functions of the code are organized and the test for each is in their own package. This made it easier to see the separation between the different portions of the program and what they do. It can be seen in the beginning lines when starting the code. For example, in ContactTest.java it can be seen as:   
package test;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import contact.Contact;

There you can see the package name in the first line and the import of Contact.java in the contact package. A thing I could have improved on is making more notes in my code. I felt like I was able to understand everything based on the names I chose for the variables, but that doesn't mean that if I were to have someone else look at it that they would have the same thought process. In the future, I should add more comments in my codes.

I ensured that the code was efficient by following exactly what the requirements were. To explain this further, I did not test anything else besides what was specifically specified. For example, I did not test a minimum length of first name and last name since all that was required was that it couldn’t be null and had a maximum length. I only followed the specified requirements. Using ContactTest.java again as an example:  
@Test

void testContactIDTooLong() {

Assertions.assertThrows(IllegalArguementException.class, () -> {

new Contact("12345111111", "Lien", "Ashley", "123456789", "123 Fake St.");

}); }

@Test

void testContactFirstNameTooLong() {

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Contact("12345", "AshleyMichelleLien", "Lien", "123456789", "123 Fake St.");

}); }

@Test

void testContactLastNameTooLong() {

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Contact("12345", "Ashley", "AshleyMichelleLien", "123456789", "123 Fake St.");

}); }

Here I am only testing the maximum length that was determined by the customer requirements.

The software testing techniques that I employed for this project were manual testing, functional testing, and boundary value testing. Manual testing only involved me doing the manual inspection of the code. Functional testing was done with the JUnit tests that involved breaking down the code and testing each requirement that was set by the customer. These tests would also be considered boundary value testing, since some of the requirements had a maximum number of characters used. For example:

@Test

void testTaskIDTooLong() {

Assertions.assertThrows(IllegalArguementException.class, () -> {

new Contact("12345111111", "HelloWorld", "DescriptionTest");

}); }

This is testing to make sure that the code verifies the length of the contact ID being too long.

Some software techniques that I did not use for the milestones were system testing and performance testing. I did not do system testing or performance testing because the customer did not create a set of specific system requirements. As far as I am aware, there are no specific hardware requirements or embedded system to worry about.

When deciding which software testing technique to use it is important to look at individual development projects or situations. I previously mentioned that I did not conduct system testing or performance testing for the milestones because I did not have any specific hardware requirements set. This would be different if the code was being used in an embedded system since then the code would have to be designed to work for a very specific specification. Manual testing should always be done by the developer since they should be testing their code during and after to make sure that it is working properly.

In acting as a software tester, I employed caution throughout the project. It is important as a tester to make sure that all requirements are followed as specified. I made sure that each requirement that was set was specifically tested. Nothing was skipped over. The complexity and interrelationships of the code I was testing was seen when each part of the code came together in the final product. Breaking down each function of the code and testing it instead of looking at it wholly was the best way to go about making sure the interrelationships and each individual piece was meeting all requirements while running properly with no errors.

I limited my bias in review of the code by looking at the code objectively. I looked at the requirements that were set and then at the code I had written and made sure that it was tested appropriately. I cannot imagine that I would hold any bias in the future if I was responsible for testing my own code as a software developer. This is because when developing code, I just follow the guidelines that are set by the customer. With the requirements and functionality of the code specified, it is easy to look at it with little bias. I believe if a project was more subjective and I had freedoms to make certain decisions, I could hold more of a bias towards it, however, I still understand the importance of testing and I do not think I would skip testing any aspects of my code.

It is very important to be disciplined in the quality of code produced as a software engineering professional. The quality can be seen during this project with the packages and their functionality, organization of code, naming conventions, and notes. These all make the code more readable, easier to update, and easier for other people to look at and understand. It is important not to cut corners when it comes to writing and testing code because then it makes for a weak product. The functionality could be off, or requirements could not be met. I can avoid technical debt as a practitioner in the field by maintaining the quality of writing and testing when coding.

Citation:

GeeksforGeeks. (2023, February 6). *Software testing techniques*. GeeksforGeeks. Retrieved April 16, 2023, from https://www.geeksforgeeks.org/software-testing-techniques/#