**Summary & Reflections Report**

**Summary**

Grand Strand Systems mobile application was created for their customers. We delivered software that included contacts, service appointments, and tasks. In this report, we will discuss the unit testing approach for three different software features. Our approach for each feature was aligned with the software requirements that were given to us by Grand Stran Systems. We have evidence to support this claim, that our code was constructed by these requirements. Our JUnit tests have been designed with a high level of quality, and we can demonstrate their effectiveness based on the coverage percentage achieved.

Writing in JUnit tests has been positive, and we have gained valuable insights into the process of creating said tests. We ensure that our code was technically sound by reviewing each line of code in detail and testing edge cases thoroughly. For instance, in most of the requirements used were “…. Shall not be null” Like the requirement “A contact ID shall not be longer than 10 characters and is not null.” We can use a line of code with a JUnit test to test that the null scenario is true. This would be ‘assertNull’

To ensure that our code was efficient, we implemented several strategies such as minimizing the use of loops. We kept the code clean and clear to what was directed from Grand Strand Systems. Something that helped keep the memory storage from having further issues is that we only allowed a unique ID for each class within the application. We also made it so that you cannot update said ID. For instance, appointment requirements state “The appointment object shall have a required unique appointment ID String that cannot be longer than 10 characters. The appointment ID shall not be null and shall not be updatable.” This helps because if you have multiple IDs the system could run an error and cause a problem for whoever is trying to book an appointment. In this JUnit test, we check that the appointment ID is not null. In the second part of the JUnit test, we check that the appointment ID is unique and not updateable. To do this we had to create two appointments with the same ID, and we ensure that their IDs are equal, but not their object references. We then try to update the ID of the appointments, and we ensure that it doesn’t change. Finally, we must check that the appointment is no longer than 10 characters, and we truncate it if it is. On the following page, you can see the outline of this code for the test cases. To show you specific evidence of how the JUnit tests work.

public class AppointmentTest {

@Test

public void appointmentId\_ShouldNotBeNull() {

Appointment appointment = new Appointment();

assertNotNull(appointment.getId());

}

@Test

public void appointmentId\_ShouldBeUniqueAndNotUpdatable() {

Appointment appointment1 = new Appointment("1234567890");

Appointment appointment2 = new Appointment("1234567890");

assertEquals("1234567890", appointment1.getId());

assertNotEquals(appointment1.getId(), appointment2.getId());

appointment1.setId("newId");

assertEquals("1234567890", appointment1.getId());

}

@Test

public void appointmentId\_ShouldNotBeLongerThan10Characters() {

Appointment appointment = new Appointment("12345678901");

assertEquals("1234567890", appointment.getId());

}

}

Our unit testing approach was thorough and effective, and we have provided specific evidence to support our claims. The coverage percentage refers to the percentage of code in a software application that has been executed during testing. We use this to measure how thoroughly the code has been tested, and it helps identify areas of the code that may not have been tested yet. Most if not all our code that was run for this web application was above 80% while testing. With further time and execution, we should be able to go back to the web application to continue testing more additions that Grand Strand Systems could ask for.

**Reflection**

**Testing Techniques**

In the Grand Strand Systems web application, we employed several different software testing techniques. The main testing technique that we focused on was unit testing. Unit testing involved testing individual code units to ensure the functionality of the code written. After we complete unit testing, we would want to move on to integration testing and system testing. I want to state that you also need to run your code as you are working on it to make sure there are no errors written in any of your classes before creating the Junit test. Just because your code has no syntax errors does not mean that it can run correctly.

Integration testing and System testing are the other software techniques that we haven’t quite gotten to use quite yet. Integration testing involves testing how the different modules work together. Meaning our next steps would be integrating our Task, Appointment, and Contact packages together, and to see if they work together without any major issues. System Testing is the second testing technique we didn’t have a chance to use which is going to involve testing the entire system to ensure it meets the requirements and specifications. Other testing techniques could also include Performance testing, regression testing, and security testing.

The practical uses and implications of these techniques vary depending on the software development project and situation. Unit testing is useful for detecting and fixing errors early in the development cycle. Integration testing helps identify any issues that may arise when modules are combined. System testing ensures that the system meets the specifications and is ready for deployment. Regression testing is useful for maintaining the quality of the system as changes are made. Performance testing is important for systems that require fast response times, and security testing is important for systems that handle sensitive data.

**Mindset**:

During the project as the software engineer on this project. I adopted a cautious mindset to ensure that I tested the code thoroughly and detected any issues before they were deployed. The complexity and interrelationships of the code I was testing ensured that I did not miss any important details. For example, when unit testing, I looked at the code coverage and how the data was being stored, and how each line of code was running. What was executing, and if it wasn’t how could I get it too?

To limit bias in my review of the code, I tried to approach the testing from different perspectives and consider different scenarios. Bias can be a concern for software developers responsible for testing their own code as they may overlook issues or assume the code works as intended without testing it thoroughly. Kind of like the remark I said above, you may think the code that you wrote works and executes well, but you won’t know this until you properly test said code.

Importance of being disciplined to commitment to quality:

Software engineering professionals must be disciplined in their commitment to quality to ensure that the software that is developed is reliable and meets all requirements. Cutting corners when writing or testing code can result in a major error, which can turn the entire system into a nightmare to maintain or update.

To avoid technical debt, software engineers should follow best practices such as writing clean and maintainable code, testing thoroughly, and reviewing code regularly. For example, during the project, we followed code standards and conventions to ensure the code was easy to read and understand, we created JUnit tests that followed the requirements given to us and were able to identify any of the potential issues before they were deployed.

Sources

Academy, C. (2016, May 24). *When coding goes wrong*. Medium. Retrieved April 16, 2023, from <https://medium.com/@coderacademy/when-coding-goes-wrong-e46d84c6565f>

Learning, C. (2002). *EBSCO Discovery Service - Academic Libraries: EBSCO*. EBSCO Information Services, Inc. | www.ebsco.com. Retrieved April 16, 2023, from https://www.ebsco.com/academic-libraries/products/ebsco-discovery-service

ReQuest. (2020, July 7). *Levels of testing - understand the difference b/W different levels & types*. ReQtest. Retrieved April 16, 2023, from <https://reqtest.com/testing-blog/differe> nt-levels-of-testing/#:~:text=There%20are%20generally%20four%20recognized,system%20testing%2C%20and%20acceptance%20testing.