Alex Casanova

CS-320 Project 2

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The approach I took for testing the application was through developing a clear understanding of the customer’s requirements. For example, each data field had specific type and length requirements. Each test I ran on the code kept these requirements in mind, and I tested for both valid and invalid inputs for each field.

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

The biggest part of my understanding of the effectiveness of my test was a thorough reading of the customer’s requirements. In this case, they were fairly simple, and could be written up in a few lines of code each. In larger, more complex projects, this may require further understanding of customer requirements, including performance testing, black box testing, and other methodologies.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was technically sound?

Writing technically sound code is essential in developing modern business applications. JUnit is a testing platform which allows you to add tests over time, allowing for the development of rigorous testing platforms for businesses over time. These tests can be applied to a number of modules, like I did for the Contact, Task and Appt classes.

* + 1. How did you ensure that your code was efficient?

While developing technically sound code is important, it’s also important to recognize that as these tests become more and more rigorous, the resources used by the tests increases. Thus, writing efficient code is also very important when writing scalable applications. I ensured that these tests were efficient by minimizing the complexity of the tests I ran. Thankfully, this application mostly relied on simple input validation, and more complex tests were not necessary.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

In each of the milestones, I have used JUnit testing. This automated testing method checks for errors when running inputs on certain code, and can return these errors to testers, including their location and the given inputs. This helps testers to inform developers of the specific conditions in which their software failed a given battery of tests. This is a very valuable tool for software development teams, as it reduces the need for manual testing of a wide variety of conditions. In addition, these testing protocols can be built on, creating large suites of tests which can be run automatically by calling the test like a method. This allows testers to run countless tests by developing a complex testing application containing many tests.

Acceptance testing is the process of ensuring that an application meets the requirements of stakeholders. These tests included ensuring that IDs could not be changed, and inputs were validated to a specific set of parameters. These parameters included making sure the fields were contained within a specific length, and that appointments could not be booked for dates in the past.

The testing done here was White Box testing, where the inner workings of the software, including source code, were known. Black box testing, on the other hand, simply puts inputs in and analyzes the outputs, without knowledge of how the code works.

This testing was Dynamic Testing, since the code was run during the tests. Static testing on the other hand, is focused more on code analysis and design.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

There are several methods for software testing that I did not use in these milestones. These methods include Integration testing, the testing of applications in conjunction with other applications. Integration testing is built on the idea of applications (or modules within individual applications) working together, and this project did not rely on this level of infrastructure. No regression testing was done during these tests. Since we are building an application from scratch, no regression from previous states has occurred, and this testing does not apply. Load testing, or the measurement of an application’s load on system resources, was also not performed. Penetration testing, the analysis of an application’s security through active attacks, was also not performed.

* 1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

Being a cautious tester is crucial in maintaining the interoperability of functions within a codebase. The main way I exercised caution when testing the application was ensuring that each collection of objects – those associated with Tasks, Contacts, and Appointments – all had their own tests, and all met the requirements specific to each one.

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

Limiting bias in the scope of this project was relatively easy, since the requirements were fairly explicit from the customer, with type and length requirements for a list of inputs - If the inputs were a certain length and type, accept them. However, on larger, more complex projects, the way these tests are laid out may introduce blind spots in testing, such as those with special characters in their names. These tests were not conducted, and may be required as the project scales.

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

One of the worst mistakes development teams can make is cutting corners, especially in testing. Failing to thoroughly test a piece of software can lead to security vulnerabilities, from invalid inputs, cross-site scripting, and other attacks. These attack methods should be used as tests, and example attacks can be used as tests to ensure they fail before a piece of software is ready for release.