Project Two

Nick Vito

CS-320

When it came to certain requirements, I think my testing aligned very well with them. Testing the Task files requirements were that each string field had a maximum allowed character number with the ID having a 10-character limit, the Name field having a 20-character limit and the description field limited to 50 characters. Using JUnit testing I assured that the code I wrote would work within the restraints of the requirements requested. I made sure to check if the code would catch an entry that broke the character limit, and it did so correctly.

The ContactTest class focuses on validating the creation of contacts and making sure that the input data falls under the requirements given. The tests ensure that the contacts phone number, name, and address are correctly saved just as they were entered into the system. The tests also confirm that leaving a particular piece of data blank throws an illegal argument. ContactService was tested by making sure the code that handled deleting, adding, and updating the contacts in the database was correct. The JUnit test confirmed that the code ran correctly during that test as well.

With the Appointment testing I made sure to test crucial validations to ensure appointment IDs, dates, and descriptions met the specific criteria that was given to me such as no field shall be null, appointment dates can not be from the past, and that they fall within the certain character limits requested. I also verified that the system will be able to update an appointment date to a future date and that a description will have the ability to be modified if needed.

Testing the functionality of AppointmentService I specifically focused on the code’s ability to add, delete, and retrieve appointments. The tests ensured that the file will be able to successfully add an appointment and retrieve appointments using their ID, which validates the integrity and retrievability of the appointments. I also made a specific test case to make sure that the code would not add a duplicate appointment under the same ID.

I am sure that my JUnit Testing was done correctly. There were a few times where it failed testing and when I updated my code to include no errors or possible spelling mistakes in the testing data it ran correctly with zero errors. Also, when looking at the coverage window most of the java files got between 85-100% coverage when running a JUnit test with their respective test classes with most classes coming in between 95-100%.

During this project JUnit testing was the primary testing technique employed. It focuses on verifying the accuracy of specific pieces of the source code, including functions or methods. Using assertions to check the state of an object after particular operations, expecting exceptions for invalid inputs, and ensuring method outputs match expected values. Junit testing is crucial in identifying bugs early in the development cycle. It is one of the most effective ways of debugging. It helps provide immediate feedback during coding.

The testing techniques that I did not use during this project would be integration testing and system testing. Integration testing focuses on combining units or components and testing them as a group to identify interface defects between modules. System testing, on the other hand, evaluates the system's compliance with the specified requirements and is often performed in an environment that mimics production. These two techniques are characterized by their broader scope of testing. Integration testing aims to catch issues that arise when components interact, and System testing aims to validate the system’s behavior.

Unit testing is the most practical testing done during the development stage of a program. It focuses on the individual logic of components before integration. It helps to improve code quality and allows for easier refactoring. This testing technique is beneficial across many software development projects, especially in agile environments where code is constantly changing and rapid feedback on the code is paramount. Integration is crucial for complex systems where different components or services must work together seamlessly. It is particularly useful in microservices architecture or when integrating third party services. Finally, system testing is essential for end-to-end testing scenarios, which ensures that the application meets all requirements. Its use is critical in release cycles, customer acceptance testing, and before deploying to production, applicable across all types of software projects to guarantee overall system quality and user satisfaction.

Adopting the mindset of a software tester required a deep appreciation of the complexity of coding and required learning more about testing code and JUnit testing as well. Also understanding the way different components interact with each other is crucial, because even minor changes could affect one area or even every area of the code. For example, ensuring that the Appointment and Contact classes correctly handle invalid input was critical, as these classes likely serve as foundational elements in a larger system where data integrity is important. Taking this complexity into account led to me running thorough tests for each functions potential edge cases.

Limiting bias in the review of the code was done by following objective criteria for code quality and functionality, such as adhering to requirements, code readability, and proper test coverage. I tried my best to make sure that when running the JUnit tests that I got the highest % of coverage for the code. Bias can definitely come into play especially when a developer is testing his own code. They could easily overlook possible bugs or make assumptions on how they want their code to function. For instance, a developer might not test a wide variety of edge cases based on the confidence of their work. Having the awareness of these potential biases underscores the importance of practices like peer review and employing automated tests.

The importance of discipline in maintaining a commitment to quality cannot be overstated in software engineering. Cutting corners in writing or testing code often leads to technical problems such as broken code or software and programs that do not function properly. Also implementing temporary solutions or rushed implementations compromises the system’s overall quality. For example, implementing code without proper testing could lead to undiscovered bugs that could cause issues and wasted time in trying to fix the bugs after implementation. Adopting a disciplined approach to coding and testing is essential to being a solid developer and avoiding technical debt.