Project Two Reflection

Starting with Contact Service I created JUnit tests to verify that the addition of contacts was successful to ensure that the service correctly stored the information. Next, I did ensure that the contacts can be deleted successfully as well and using the JUnit testing to validate the proper function of the delete contact method. Finally, I made sure to validate the proper adding, deleting, and updating as well as the JUnit testing validating the null and character limit. Contact service aligns with the software requirements by “the contact object shall have a required unique contact ID String that cannot be longer than 10 characters.” My test for unique IDs aligns with this by not allowing duplicate IDS cannot be added.

Task Service was also testing the addition and deletion and I confirmed the accuracy as well as assessing the correctness of the add and delete task methods. The JUnit testing ensured the task’s name and descriptions fields being updated correctly, this validated the updateTask and updateTaskDescription, this also tested that the unique IDS were indeed unique upon addition. Task Service aligns with the software requirements by having “the task object shall have a required unique task ID String that cannot be longer than 10 characters.” My testing confirmed that the unique IDS aligns with the requirement ensuring the tasks are unique.

In Appointment Service I created tests to validate the addition and deletion of appointments, I also did tests for date validations to ensure that appointments in the past dates were rejected, so I had to make sure the dates were in the future. I also had to ensure that the appointments were assigned unique IDS. Appointment Services align with the software requirements “the appointment object shall have a required unique appointment ID string that cannot be longer than 10 characters.” The testing for unique appointment IDs meets this requirement.

The overall quality of my JUnit tests has lots of test coverage, it achieves coverage by testing all the various scenarios within each method, the tests helped finalize and ensure it was error-free. I also had effective assertions that were able to validate expected outcomes, for example using the method assertEquals was able to be used to check if a contact, task, or appointment was added successfully. My software also was able to handle exceptions by ensuring that the correct exceptions were thrown indicates the effectiveness of tests. I feel that the overall coverage was achieved.

When writing my JUnit tests, it was a lot of trial and error, I had to really go into depth on what I was testing and making sure that they were giving me 100% when running the tests. When talking about validations I was able to create a piece that validated the use of making sure the code was non-null and had a unique ID, to ensure this I was using “IllegalArgumentException”, I used this for contact service, task service, and appointment service.

When I was in Contact Service, I was creating tests in the tests portion that ensured my code was efficient, for example when testing I used testing isolation to ensure that adding a contact was successful, I used public void testAddContactSuccessfully and so on for this specific code piece. I also used focused assertions to ensure the efficiency of my code by using “assertEquals” and showcasing that the test was working.

The testing techniques that I employed in this project were Unit testing, and unit testing consisted of testing individual units or components of a system in isolation, my JUnit was used for this purpose. The practical uses were to verify that the individual components functioned as expected. I also used test isolation as mentioned above to ensure that each of my tests were independent. Examples are when adding, deleting, and updating contacts, they were all individual tests.

Thinking of the techniques that were not used I would say I didn’t use integration testing and the reason for this is because in smaller projects, integration testing is used in larger and more critical software. Another technique I didn’t use was performance testing and because I didn’t use was because it is often used for production-ready applications and this specific software did not have this.

When approaching this project as a software tester, I had a methodical mindset, I was able to embrace a cautious mindset to ensure I was able to mitigate risks, uncover hidden issues, and make sure the code was reliable. An example of the method that I used was testing involved user input, I designed cases for boundary values and potential invalid inputs.

I feel I tried to limit bias in my code review by adherence to standards and guidelines, for this I focused on using the best practices and following the guidelines and an example would be following the guidelines listed for me instead of doing my own. Another way I avoided being bias by thinking about the code base and its merits, this helps prevent biases related to personal relationships. When being biased in self-testing I had to look out for confirmation bias, as well as overlooking alternative approaches, and underestimating error proneness, I tried to strive for a balanced code review.

It is very important to be disciplined in my commitment to quality as an engineering professional by writing high-quality and well documented code that ensures it will be maintainable over the long term. Cutting corners can heavily affect the quality of your code and cause it to malfunction in a way that can be a huge loss, I feel it is so important to always follow the guidelines and test over and over to ensure the quality is going to withstand. I am going to ensure that I avoid technical debt as a practitioner in the field by following the best practices, having regular code review, use automated testing, always continue to learn, and plan everything. I know that these core strategies will contribute to the creation of high-quality software in the future.