Darius Quick

9/29/23

Project 2 Essay

When creating my code, I made sure that all of the functional requirements including the variables needed, the lengths restrictions, other restrictions, and the ability to add, delete, and update contacts, tasks, and appointments were all included. The tests that were to be created to test each function for adding, deleting, or updating a contact, task, or appointment as well as to test the restrictions placed on each variable were also included. My Junit Tests were effective because all of my coverage results for my test modules were above 90 percent. And if the tests coverage was high as long as the contact, contactService, task, taskService, appointment, and appointmentService modules met correct functionality then the program is set the way it needs to be.

I made sure my code was technically sound by following functional requirements of the project. For example all of the variables under contact, tasks, and appointments except for appointmenDate have restrictions under exceptions that only allow them to be a certain length and they cannot be ‘null’. I also know that my code is technically sound because I reviewed my code for the Service modules to make sure that they were completely accurate, the tests from the ServiceTest modules worked as directed and the Junit Tests passed with over 90 percent accuracy.

For Project one I used many different software testing techniques for each part of the assignmnent. For the contact modules black box testing techniques were used to improve my coverage and tests. I mostly used checklists and informal reviews in order to see if it met functional requirements and was accurate. If I found a defect, I would search for the source then correct it. For the Task modules I had to reach out to a 24-7 tutor to show me how to use J-Unit to test the coverage and accuracy of my code. The tutor was extremely effective in teaching me how to use Junit. After his lessons, I was able to go back and improve the contact, contactService, contactTest, and contactServiceTest classes. After using both what I had learned of informal review white-box testing techniques, and the code from the contact modules (Experience based testing) I was easily able to create the task, taskService, taskTest, and taskServiceTest modules. Experience based testing from the Contact and Task Modules was how I was able to start working on the Appointment modules but there were parts that I had to apply new skills like the ‘Date’ data type that was applied to the variable appointmentDate. This made me have to make some changes to my code. Creating the tests was also different from the contact and task modules.

The testing techniques that I did not use were Boundary value analysis, walkthroughs, technical reviews, or inspections. Boundary Value Analysis testing techniques tests the boundary of 3 numbers at the end, middle and beginning of a span of numbers in the code. The white box review techniques after informal review were simply unavailable to me so I could not use them.

When you are doing a project that needs a large amount of people to do or for a large project White Box testing techniques are very effective because the there are plenty of people to review your work and give you different perspectives. The reviewer would be able to catch an error or defect that you would not because he did not do the work. Using Black Box testing can help to make sure your code is efficient. Using Junit has taught me to not only make sure my code is accurate and functions but provides the correct coverage when testing it. State Transition Testing was used to ensure that my functions to add, update, or delete a task, appointment, or contact test. This was useful in understanding how this testing technique is effective in testing your methods.

It took changing my mindset from building successful code to making sure that the code is accurate by trying to prove the code doesn’t work in order to complete this assignment. I did have to exercise caution by first making sure that my code was accurate because the tests are useless if the code isn’t technically sound. I then had to exercise caution by creating tests that accurately test the function of each method and that the restrictions on the variables work as intended. I believe it’s more important to understand the complexity and interrelationships of the code than to appreciate them because it helps you create the tests properly rather than just admiring the code that was created. I limited bias by making sure to test only once for each specific purpose. If I were testing my own code, I can see how this would be difficult if it were a larger program because the larger the program the more tests that would be needed.

It is important to be disciplined in my commitment to quality as a software engineering professional because this creates programs that work. And the overall mission of any project I should be working on is for the program to work and function as the user wanted me to create it to do. To produce anything else is contrary to the very objective. Which is why you should not cut corners and try to produce code that is efficient. I plan to avoid technical debt as a practitioner by creating code that works and if I am unable to do so I plan to learn more until I am able to do so. I will commit myself to this.