**Summary and Reflection Report**

For the customers mobile application, I used Junit testing for contact services, task services, and appointment services to meet the customers requirements. To ensure that the software requirements were aligned I made sure to read the requirements and followed them to write my code. For example, I ensured that the customer contact ID was no longer than 10 characters, and couldn’t be null and updatable. I did have some difficulty creating the tests for the three milestones like adding, deleting, and updating but the other classes were easy and simple to write.

To ensure the overall quality of the Junit tests, I used combinations of tests. I ran my code in small units which made it easier for me to fix any errors as I wrote. To ensure that the Junit tests were effective, I made sure that my code was covering over 80 percent. I struggled the most with the task milestone than the contact and appointment services.

I ensure that my code is technically sound I used variety of assertions to run my tests. Using assertions helps you understand your code better. It helps you know where the code has errors and needs fixing in able for the code to produce the desired outcome.

Assertions.assertEquals(Task, TaskService.getTask(“123”));

I ensured that the code was efficient by using proper variables and repeated tests. Proper variables that meet the customers requirement can make the code easier to read and maintain . A lot of the classes and tests for the projects were similar so I didn’t have to change much which can help save a lot of time writing code and fixing errors.

contact contact = new contact("1234","Jessica","Bains","1234567890","798 Witch Lane");

assertTrue(contact.getid().equals("1234"));

assertTrue(contact.getfirstName().equals("Jessica"));

assertTrue(contact.getlastName().equals("Bains"));

assertTrue(contact.getphone().equals("1234567890"));

assertTrue(contact.getaddress().equals("798 Witch Lane"));

I used Unit testing for all three features by testing smaller parts of the codes, units, to free the code from any errors and make sure the code is working as expected. I also used integration testing all in the milestones, it helped me determine how well the smaller units of codes work together and if the system is working properly. I also used system testing, this technique helped me examine every part of the code to make sure that the software application runs properly as a whole.

The other software testing technique that I did not use is acceptance testing. Acceptance testing technique can help determine if the product meets the users requirements. This test can help improve the product since we are making the product for the users. Another testing technique that we didn’t use is performance testing, performance testing technique can help determine the speed, scalability, and stability under the expected workload for the system. This test helps the product meet performance testing goals and provide good user experience. Another testing that I didn’t use is white-box testing. White-box testing is testing of the internal coding of the software system. For this you need to have knowledge of the internal structure and code of the system.

All these techniques can help create a successful and quality product by helping us find flaws in the software application. Finding flaws helps us know where exactly in the software application needs adjustment so we can prevent errors from reaching to the users.

As working as a software tester, I learned through all the modules that it is important to be cautious and precisive while writing your code. It is important to understand the customers requirement and keep that in mind while creating the product, and to make the project easier to read and organized so it is easier to maintain and make any changes if needed. The complexity and the interrelationships of the code has more chances of unintentional errors so it is important to keep this in mind as well while writing your code.

As the software developer, avoiding bias is something you need to be aware of especially when you are responsible for testing your own code because sometimes we can’t see errors in our own code. When testing your codes sometimes you may bring unconscious and unintentional bias so it is important to keep that in mind and to not misunderstand the customers’ requirements in order to create a successful desired outcome.

As a software developer professional, being disciplined in your commitment to quality which means not to cut corners when it comes to writing or testing your code. Committing to quality work can ensure that your software is reliable and meets all the customers requirements. This developer quality can help prevents defects in the code that can cause major issues in the future. This can also prevent rework which can costs a lot of money and time, and loss of reputation.