**Project 2: Summary and reflection**

Eric Breznen

Southern New Hampshire University

CS-320-R4835 Software Test Automation& QA 24EW4

Professor Ramsey Kraya

04/17/2024

My approach to testing was to ensure all the requirements were met. For each project, I started with the object class (Task, Appointment, and Contact). I had the Eclipse IDE up on one screen and the requirements on the other to ensure that each object class had the appropriate attributes, getters, and setters. Once the objects had what were necessary, I moved on to the service classes. These were more difficult and, in their first iterations had the user interface, they printed out menus, took input, and performed the functions. Then, in order to aid testing, I removed all of that so that the classes only had the methods, functions, and variables that were necessary. Each of them had an ArrayList that stored the objects after they were created, methods to create new objects, delete existing objects, and update all object attributes except for the IDs, since the requirements stated that the IDs could not be updatable, the attributes in the classes were marked final and there was no functionality in either class that could update them, in fact, they did not take input either. The ID generation was completely self-contained within the service class. Whenever a new class was created, all attributes except for the ID were taken as input, the ID was generated based on the previous ID that was put through the GenID() method that took the string number ID, turned it into an integer, incremented it, turned it back into a string and returned it as the new unique ID for the new object. An added benefit of this is that when all three service classes were combined into a single program through a main class, the unique IDs would match up between the different objects which could assist with the functionality of the program as a whole assuming that all three were created at the same time. Otherwise, another method would need to be added to the appointment and task methods to take in and store the ID of the appropriate contact.

I wrote each test to make sure that each function worked properly, I wrote multiple tests for each class to test data validation to ensure that none of the attributes of a fully created object were null or were too long. Each test created objects and then used the methods within the service or object class to test various functions that existed as dictated by the requirements. The testing coverage shows that the service classes and object classes were tested thoroughly. The highest coverage percentage was the AppointmentServices project at 89.5% where the lowest tested functionality was the GenID method at 76%. This is due to the fact that for it to test the uniqueness net, more than 999,999,999 objects would have needed to be created and stored. Confidence in the fact that there would not be that many appointments, tasks, or contacts (as a similar method was used to generate the ID for each object) was the confidence in uniqueness of IDs. The next highest was the TaskServices project at 87.3% followed by ContactServices at 66.7%. The reason this one was so low was that I wrote functions into the service class to assist with the testing that weren’t necessary to test and I chose to focus on the methods that were required.

The JUnit tests were difficult for me to learn. I failed all three milestones and committed to significantly improving my testing knowledge and capability. I wanted to show that I could effectively test every method, so I was meticulous in ensuring that each required attribute and method was tested for data validation, exception throwing, and successful functionality. The idea was to follow the style of the Task test where it first created 1 object, made sure that each attribute was as it was input and tested for validation. Then the test created an object, and attempted to update the attributes with invalid input to test whether an illegal argument exception was thrown. The Service test classes all created one object, tested to make sure the ArrayList had the correct number of objects, deleted the object, retested the length of the ArrayList, and then attempted to create an object with invalid input to ensure the proper exception was thrown. The second test created multiple objects, made sure they were added to the ArrayList appropriately and then deleted an object to ensure that that functionality requirement was met. The last test created an object and then tested all updatables to ensure that the correct object was called from the ArrayList and that that object was updated appropriately.

I wanted to ensure that my applications met all of the requirements, I used mostly automated testing, functional testing, unit testing, integration testing, and acceptance testing techniques. The automated testing came with the JUnit tests, I didn’t manually test my final programs, but I did in an earlier version that had the user interface, once that was removed there was little way to manually test as there was no displayed output that I could check. I tested each function for expected behavior through the automated testing, making sure that each method was tested for both valid and invalid input and correctly updated. The integration testing was performed by connecting each object class with their respective service class to ensure that they worked together appropriately as they are expected to. Unit testing was done when I tested each method meticulously to ensure that it could handle both valid and invalid input and if it called or was a setter that it did what it was expected to do. In a way I used regression testing in the multiple iterations of the programs to make sure that when pieces were removed (namely the user interface) or updated (all of them) no new issues were introduced and the programs were only improved. With the lack of user interface, the usability testing wasn’t necessary, the functionality was the important piece to cover.

In the beginning, starting with milestone 1, I didn’t exercise enough caution, I was reckless and careless, and the results speak for themselves. As the pipeline moved on I developed a sense of more and more caution, being extra careful to make sure that each piece of the program was tested carefully for both valid and invalid inputs and making sure that everything was covered. My discipline also lacked towards the beginning of the project development process. I failed to look at resources, at announcements, at things that would have helped. As the project progressed, I started using more and more resources, and committed to ensuring the quality of my testing and of my software, to make sure that it all acted as it should. Taking pride in one’s work is important, however removing bias is also important, it is easy to assume, when working with your own code, that you are correct. It is difficult but a good method is to assume you are wrong when writing code, so that when you are testing you are checking for and finding those errors. It all comes down to taking pride in your work so that you can get it as good and correct as possible, after all, your name is going on it, it is one piece of your legacy. I lacked that this course, especially with the milestones. I believe I got better in project 1. I took the time to make sure that it was as correct as I could get it. I still lacked in areas but the improvement, I think, speaks for itself.