Project Two

Joshua Norris

Southern New Hampshire University

Project Two

Working with JUnit tests and the contact, contact service, task, and task service classes was a challenging project. Because I was also reacquainting myself with how to write Java code while working on the assignment, I first fell into making the contact and contact service classes using the given requirements before working on the testing classes. I realized that this could lead to trouble later when I had to rewrite parts of the contact service class to ensure it passed all tests. I believe the first example of this is ensuring that a given name does not exceed ten characters when updating the name. I can see now why testing and writing the code should be in unison, and how testing can reveal oversites in code.

I believe my JUnit tests each requirement given for the different classes. On some tasks, such as deleting tasks, I have achieved 100% coverage. However, there are other methods that do not cover all areas. Once example is the Contact class when running the Contact Service Test. I am unsure how to reach 100% on this portion of the test, or of what impact the missing coverage could have.

I ensured my code was technically sound by at first creating a main method and running the code with print statements to verify it did everything that I wanted it to. This was while I was still grasping how to complete the assignment and meet requirements. Reviewing the milestone guidance, I realized directions said to use JUnit tests to verify functionality. So, I then removed the main method and worked on the JUnit tests to see what was covered and what was missing. One example of this was testing if a contact name was too long. I did not originally include such a test and added after this.

I will admit that I feel like the efficiency of my code is still lacking. However, I made changes when working on the Task Service class to fix what I found inefficient about my previous code in the Contact Service class. One example of this was that most of my classes required the task list to be provided as a parameter, and then returned the task list once it had been updated or created. I found this to be unnecessary and removed it from the task service class. The resulting code at least looks cleaner and more efficient. I would still love to have my code reviewed and any input on how to make it more efficient given.

Looking at the different test levels of the V-model, I have mostly stuck with unit testing throughout the milestones. Unit testing involves taking the different components of code that has been written and testing to ensure it meets functionality before integrating with other components (Samaroo, 2015, p. 46). Examples of this in my code are the class tests in JUnit, such as TaskTest.java. The only integration testing, the next level of testing in the V-model, would be the service class tests. However, these only test integration between the service and the respective class, such as appointment and appointment service. The service classes rely on the separate class programs to function correctly, so tests such as appointmentServiceTest.java effectively tests that functionality. To me, this is not true integration testing because I have not integrated all components into one system.

Along with integration testing, I have not used system testing or acceptance testing. System testing involves testing of functionality from an end-to-end perspective (Samaroo, 2015, p. 49). Since I have not gotten to that stage of the V-model, and do not have all units of the program created yet, I cannot do this test. This also leads to acceptance testing, where the program would be tested to ensure it meets user needs (Samaroo, 2015, p. 51). This would of course happen after system testing, even though the requirements would have been hashed out toward the beginning of drafting requirements.

Of these four levels of testing, each serves a purpose in its own area. Unit testing is core to every bit of code written and is needed to ensure it functions as intended. Moving to the next level, or integration testing, you will need to make sure all of you previously tested units work well together. They need to integrate without bugs or other issues. Third, the system should be tested to find concerns that might not be apparent with the previous tests. Finally, acceptance testing to ensure you are giving the user what they needed. All of these are essential to all software projects. Doing them in this order can help solve problems earlier in development life cycle and prevent mistakes from becoming too costly to fix.

While working on the project I often found that there were parts of the code that were not tested. I had to go back many times to address the little “nooks and crannies” of the program that were originally missed. These were usually due to how the service class and object class interacted with each other. I had to use caution in later portions, such as in the task service, to ensure everything was covered. These areas included testing each possible error related to parameters. A name or ID being greater than ten characters is a good example of this.

I tried to step back from my code when reviewing it to hopefully catch what I might have already missed. I was very tempted to write a user interface program to run everything and test it that way but decided it would not meet the intent of the tests. I understand that there is always a natural bias and that the only way to get a truly unbiased review of the code would be to bring in an outside observer.

I previously talked about the different test levels in the V-model. Each level builds upon the last. Therefore, discipline is required to maintain high quality code. If the original code is created with low quality and allowed to pass without thorough testing, then the code will be moved to the next level that could cause costly repairs later. This can be hard to avoid. Life will bring about the surprises that you cannot prepare for. Deadlines will sneak up on you. But the more I strive to create good code from the start, and test it thoroughly at each step, these surprises should have minimal impact to overall quality.

References

Samaroo, A. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide* (Vol. 3). BCS The Chartered Institute for IT.