Garrett Waag

Final 7-2

CS320

My tests aligned to the software requirements by making sure every requirement was addressed. I created tests to check that all the ids, names and other fields were not too long, and to test that they were not null. I created tests that used the service classes to create update and remove tasks and contacts. I also ran tests to make sure that variables and lists were being created correctly and utilized the way I wanted them to be.

The effectiveness of my Junit tests for the contact service and task service was shown by having 100% coverage for both the service and their associated class (task, contact). Showing that the service could go through every line of the task and contacts fulfilled the requirements that were needed for the software of being able to create update and remove.

To make sure that my code was technically sound I tested to make sure that variables had the correct data after populating them. One of the examples of this was testing to make sure the name variable was what I entered with. assertTrue(task.GetName().equals("testname")); I did this with all the variables. This was also used to make sure the list was being populated with tasks with assertTrue(service.GetTasks().size() == 2 I then used assertTrue(task.GetName().equals("Garrett")); to make sure that the list was being populated with the correct information.

Similar to how my code was technically sound I feel that my code was efficient because it was very basic, this allowed me to use the exact same setup and tests for both the tasks and the contacts as well as both services. The three examples above were used exactly the same in the contact with only a change from task.GetName to contact.GetName, to help myself clarify what I was working on.

For all of the modules I used the exact same types of testing. The Junit tests were identical each week. I started off each module with unit testing making sure that each class worked fully by itself. I then moved into some integration testing with the service classes since they used their respective data classes in the service. This allowed the data classes to get extra testing since they were used in multiple ways. Using the coverage test feature it allowed me to make sure that the entire data classes were ran through when using the service classes as well, showing good integration. Both of these types of testing would be considered functional testing, that check if the code itself works. The unit and integration tests both allowed me to start testing the overall software against the design document making sure all requirements that were given have been met.

There are many techniques that were not used for these modules. The most obvious would be things like system, acceptance, performance or even security testing. Since most of thee other types of testing are done after the project if farther along it would have been hard to use them. System testing could start to be used since the three milestones are now done and the entire thing could be put together. Since the prompt was given to us testing our design document against the customers’ requirements was not needed.

All of the different techniques play an important role in software development. When a project is just starting out unit tests like Junit are the most practical. They allow the developer to test the code as it is written to help keep each section the most error free. This saves time and money down the line when the project is in later stages by finding errors faster. Integration testing also should be done pretty early. Once 2 units that use each other are done it should start since the units could be perfectly written and they all work alone but have issues communicating. This is more important with multiple developers since everyone thinks a bit different. Some of the techniques that were not used require multiple people. If I was developing some software for myself then I would not have to check for customer requirements for example. Since our project does not have any sort of UI things like usability testing were not done, but in any other situation it would have been done.

When testing the code, I did not really employ any caution. My goal was to either see it work totally or break it the worst I could, to know what needed to be accounted for. I feel that it being my code that I was testing did make me a bit bias when it came to how it was done since its hard to tear your own work down sometimes even if it doesn’t work. The importance of being disciplined about quality goes farther that just being paid to make something that works. We have all used a program that may have cut some corners on quality and seen first hand how this can effect the end user. Making sure things are done the right way the first time instead of quick cheap fixes is super important. I know I have made the mistake of this many times and then have even forgotten to go back and fix it. This made me have to take extra time outside of what was originally scheduled to make the fix.