Kenneth Cluck III

04/14/2023

CS-320-T4208 Project Two

Testing is one of the most important processes in the software development lifecycle. Not only does it get an entire phase, but some sort of testing occurs at every stage of the development process. Throughout the lifecycle, there is static testing, where logic, design documents, and the systems requirements are checked. There is also dynamic testing, where programmed tests execute code, and then check the results. Testing allows us to be certain that the program requirements and expectations are met, while also ensuring that the logic of our program executes correctly. This results in a well-functioning product, as well as makes it easier to ensure our code works with the code written by other developers. Without testing, proper large-scale projects, involving teams of developers, would be nearly impossible.

**Summary**

There are a lot of different techniques used in testing. These techniques make it easier to ensure you have properly tested the software. This project was not a very large nor complex program. With this project I still used the standard testing procedures, but at a much smaller scale. My first step was to read over the directions. This was to ensure I knew what requirements had to be met. Aside from requirements it also let me know exactly what needs to be delivered. During the first milestone I had made a mistake here and written the program in a much different way. This resulted in a working program, with different deliverables. I had to rewrite that portion of the project. If I had properly reviewed the requirements and deliverables, I would have saved myself a lot of work. I did not make this mistake again. Each field had their own requirements, such as IDs being no more than 10 characters, and would need to be unique when being entered into a service. I also ensured all deliverable files were completed.

In order to make sure my software was technically sound, I made sure I met proper coverage requirements, as well as manually reviewing the project’s logic. I ran the Junit Coverage in Eclipse and found that I had missed some areas. This tool will tell you what logic has been executed during tests, and show colors based on whether each possibility was accounted for. At one point I had forgotten the tests to ensure invalid Contacts information could not be updated (ContactText.java lines 179-204), and this proved useful as I added the tests and rechecked the coverage. I also had to ensure that 80% of each object was covered, as this was in the project requirements. I exceeded these requirements in all files. This allowed me to ensure that everything needed was tested and that the program was in fact functioning as expected. This was all the more important as we were creating services without an actual interface to use these. I also managed to make the program more efficient with the use of exceptions for invalid information updates. There were several options for this, but I felt exceptions would be the simplest, easiest, and most efficient way to implement this. For example, in “Contact.java” in likes 78-83 and lines 96-109, without using exceptions, these code blocks could have been a little more complex.

**Reflection**

After I had reviewed the project requirements, I also reviewed what type of logic was needed for the project. We were to write a contacts, appointments, and tasks objects. Each field also had certain requirements. I also had to write service objects to manage these other objects. There are several ways to do this. I had to decide how the service would manage these objects as well as what sort of data structure to use. In this case I had used hash maps for all services. I also had to decide how error handling would be performed, in a way that would also be easy to test. One option would be to just alert the user that the input does not meet the requirements. The option I chose was to throw an exception, as well as storing the exception message in a static string variable. Exceptions made detecting and checking errors in the tests much simpler, and I can simply import the static string variable to compare exception messages. The review of the project’s requirements and deliverables, as well as reviewing of the project logic, is considered part of the static testing process. This will occur at the start and at the end of the project.

I decided for simplicity to write the required objects and services first, then write the tests. Another option would be to write the tests first, in a test-driven approach. However, in this case I knew the tests could be considerably larger than the program and it would be simpler to write the program logic first.

In each object and service, I reviewed each method and decision point that I had created. This included a helper method in the “Contacts” object that validated phone numbers. This also included if statements to ensure requirements are met for any input information. I wrote tests for each method and for each possible decision. I also had to ensure that the errors received were correct. I stored each exception thrown in an Exception object and imported the static string variable storing the error message. When checking exceptions, I made sure the correct type of exception was thrown, as well as the proper message.

During the project I started to ensure a more cautious mindset. Early on in the class, during the first milestone, I already had to recreate the assignment due to not paying enough attention to the project requirements. During this project, I read the requirements several times throughout the development to ensure I was not straying from the requirements. When checking coverage, I also had to be objective from a testers point of view. While I personally thought my logic was fine, there were testing requirements that had to be met and I had to make changes during this process. If I had not taken the role as a tester seriously, it would have resulted in a sub-bar application and tests. The bias of being a developer is something to suppress. If we are not disciplined in this, we could create software that either does not meet requirements or makes the jobs of other developers more difficult. Not completing requirements, by being objective, could also increase technical debt. Some items may not be completed properly and have to be added back to the backlog, making more work in the future. For example, in my milestone one I did not actually check the exception error messages. For project one I had to go back and implement this. It would have been easier for me to implement this from the start, as refactoring code is generally more difficult than writing it correctly from the start. If I had ensured proper test coverage from the beginning, I would have saved myself work in the long run. This is even more important when working on a team with other developers, that may have to finish something that I had started.