Kevin Raddatz

CS320

Project Two

08/15/2021

For each service application (Appointment, contact, and task), my approach for developing unit tests was aligned to the requirements by developing tests that tested things specifically asked for in the requirements. This was important as it ensured that the software requirements were being met by the final software application. When looking at the tests developed, it is clear that the intention of each test was to check the software for a different piece of the software requirements.

In my opinion, I think my JUnit testing could use some work. While I was able to make some good progress on my JUnit tests, I did struggle with writing some of my test cases. In the future, this is something that I would like to improve on. Unit testing (as well as using testing tools such as JUnit) will be incredibly important going forward in my career as a software developer, and some skills that I could do with brushing up on going forward.

The main method of testing for this application was without a doubt, unit testing. Unit tests were developed throughout the project to test for each portion of the software’s requirements as well as test each different file across the application (there were two files for each service) individually. Unit testing is a huge asset to software development as it can be automated (Such as the JUnit testing used in this project) making the process quicker and more efficient. It was also important to perform manual inspection of the code as a method of testing in order to make sure that the code was sound and free of errors. This is a relatively quick and easy way to make sure a piece of software is working as it should.

As for software testing methods that weren’t used, there wasn’t really any integration testing of the application as a whole. Integration testing tests the application as it is integrated together. In this instance, it would have tested how each service worked together and verified that it was integrated in a way that doesn’t cause errors. This is an incredibly important method of testing. It is possible that one of the smaller service applications could be free of errors, but when combined with other ones, could cause major issues for the application. Other important methods not used would be acceptance testing (where the completed application is tested against the customers requirements) and system testing (where the completed application is tested as a system. This would have required the application to be fully integrated).

All of these testing methods mentioned are incredibly important, but they all have different applications for different kinds of projects. For example, a simple console application, consisting of a single Java file, will not require any form of integration testing. This is because there is nothing for this piece of software to be ‘integrated’ with. Testing methods for an application of this type would consist of unit testing (Even if it has just one ‘unit’), acceptance testing, and system testing. This is in contrast to a multi-file Java application that combines multiple pieces of software to create a final application. This would require extensive use of integration testing to make sure that every different piece of code developed can be combined together to create an error free application.

When working on this project, I found that I had to put myself into a very interesting mindset. It was incredibly important to change my state of mind to focus on requirement. In my opinion, the requirements are a huge asset when testing software as it gives us an outline for what the final application should look like. I was very cautious when developing this software, quite possibly more than ever before when developing an application, as I knew that there was a chance that a mistake in one file could have an impact on the code in another file. For example, it was important to consider the relationships between the Class and service file in each different service (Task, Appointment, and contact) in order to make sure that each piece worked correctly without messing anything up.

When acting as a tester for your own code, bias is definitely a huge concern. I’ve made the mistake in the past of thinking that I can’t make ridiculous mistakes, whether it be missing semicolons or missing parentheses (or any other little mistake that causes hours of frustration) or invalid syntax in my code. I have instead attempted to take the approach of assuming that everything I do is incorrect, in order to keep myself watching for things that could cause major issues. While this may sound harsh, it is important to question even your own work in order to not be blinded by the bias or the belief that mistakes can’t be made.

When it comes to software testing, discipline is important as it will ensure a level of quality in the work that is produced. Cutting corners when writing or testing code can have severe consequences. These can cause software failure, which can have an even bigger impact, even causing safety issues depending on what the software is being used for. Developers have a responsibility to write code that is sound and that will operate according to a customer’s requirements.

# References

Hambling, B. M., Samaroo, P., Thompson, A., & Geoff Williams, P. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT.