Taya Zaragoza

Professor Krava

CS-320 Software Test, Automation QA

23 June 2024

Project Two: Reflection

When creating the code for each of these three features, I made sure that each software requirement was met. I first created each class to be sure the requirements were met. Then I created the matching test class for each. I believe the overall quality of my tests were adequate. I made sure to hit every requirement with each of the three features. For a few of the classes, I had to add more code to test the boundaries of each input component. This was an important feature that I was lacking in before. I am still quite new to Junit tests, so it is entirely possible there was something I missed when testing the my code. If my calculations are correct for the coverage percentage of my testing code, then my percentage is 52%. This is a surprisingly low percentage as you want to shoot for around 80% . This shows me that there is more I could have tested within my code.

To ensure that my code was technically sound, I used the tools that our coding software offers and ensured that there were no errors. An example of this is with my getTask function. At first I didn’t have any arguments listed and the software threw me an error. I was able to add the String argument and have no further technical errors. This same tool was used throughout the creation of all of my classes.

To ensure that my code was efficient, I made sure to write it and edit it in a way that made sense and would have the tests run smoothly. Another specific example is with my “check if null” tests. At first, I was adding a component to my array that had each variable set to null. But I was shown that the more efficient to test for null variables is to have only that one variable you are testing to be null. For example, in the Contact list, if I wanted to test if the FirstName was null, then I would need to set that variable to null but give each of the other variables their own value.

For this project, we specifically used JUnit testing in order to verify that the code created met the requirements outlined in each of the three features, as well as to test the in-memory data structures. One of the techniques used was white-box testing. This technique is used when the tester has access to the source code of what is being tested and has a complete understanding of the program. Since we had to create our own source code to test for each of the milestones, this gave us complete knowledge of the programs and allowed us to practice white-box testing.

There are many techniques when it comes to software testing, which means there are many we did not use for these milestones. The most obvious one is the opposite of white-box testing, black-box testing. This technique is when the tester does not have full knowledge of the program and tests the code without having access to the source code. There are also many different kinds of software testing platforms that we have not used, such as Selenium. However, this platform is mainly used to test web applications, so it may not have been useful in the milestones.

White-box testing is practical to use when unit testing, which tests separate sections of code. It is also best to use when testing very complicated logic within the code. This allows testers to ensure that the conditions and pathways of the code are being met against the requirements. Black-box testing is practical to use when testing functionality of a program, or with use case testing. It allows testers to ensure that the code is performing the way users would expect it to, even if they don’t know exactly how the inner workings of the code behave.

A specific example of when to use each technique in a project could be when testing a mobile application. White-box testing would be used to ensure that the logic of the application is precise and accurate. Black-box testing would be used to ensure that the application is user-friendly.

Whenever I am developing the code for any project, I always use a decent amount of caution. I believe the biggest priority when creating a project is ensuring that all the software requirements and behavioral requirements of the code are met. For projects like these, I always split my screen with one side being the developing code, and the other is the list of requirements that must be met. This ensures that I am always creating the project in the way it is wanted. With testing, I do the exact same, except you must translate the specifics of what is to be tested with the wanted outcome that is detailed. This makes the relationship between the original code and the testing code much more complex because you have to be thoughtful in how to achieve requirements and be aligned in how you test the requirements.

While using software developers to test their own code can be seen as beneficial, as they have a full understanding of the code and requirements, I feel that there is always a level of bias that developers cannot help but have. It is hard not to be prideful and confident in the program that you created, however, there must be a level of humility and understanding that human error exists. Especially with how specific you need to be when programming, it is important to do thorough and unbiased testing. Being a student and being new to coding and even newer to testing helped me keep my bias in check. However, looking back, I realize now that there are some components that I missed in testing because I was biased with my code. An example is input validation and ensuring every ID of the three features are unique.

Being disciplined as a software engineer is one of the most important things. With the new age of growing technology and data being a form of currency, software engineers have a responsibility to ensure quality with every piece of code. Not only is this discipline for the sake of security to users and all those affected by your software, but also to the company to ensure a professional and low-cost environment. Cutting corners leads to long term issues and delays. To avoid technical debt, I will do my best to keep my code efficient, descriptive, and well developed to the fullest.