Eric Boilard

Southern New Hampshire University

Project 2 Summary and Reflection

June 18th, 2021

**To what extent was your testing approach aligned to the software requirements? Support your claims with specific evidence.**

The software requirements were very specific in what needed to be included and the parameters in which they needed to follow. My testing approach was modular and incremental to follow the test design process. As one part of the code was completed in enough to run a test, I simply created the test class and ran that part of the code. When that code failed or had errors, I was able to adjust the code as I went along troubleshooting. Not only did this help me include all the requirements for the software but it also helped me make sure that they were properly being tested as each section of code was run as its own test.

**Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

The quality of my JUnit tests is good. Although it is efficient in getting the job done regarding testing all the requirements, I am sure there are areas in which I could have been more thorough and coded sections more simply. The tests themselves cover all the bases including all errors and all passing values. My successfully completing all these tests I know that the coverage percentage of my tests is very high and there are likely few things that I missed. Running each test incrementally was essential and making sure that tests ran smoothly as creating all the tests after the facts makes it difficult to build tests that run well, although it can be done, it's just not preferred.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

To make sure that my code was technically sound I made sure that a followed similar structure throughout my code blocks. For example, in both the contact class and the task class files I made sure I followed a basic structure that was easy to read and easy to change if needed. I started with simply declaring the class and the variables to be used within the class, built my constructor methods, and defined my getters and setters. Within my setters were the criteria for passing and failing which including a throw for anything too long and anything that was null. Following this structure and commenting on it made for a part of code that is sound and easily readable even to someone new to coding like myself. Additionally, I included variables with proper naming conventions and made sure I indented everything appropriately.

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

To make sure that my code was efficient I cut down on bulky sections of code that were passing their unit tests. For example, with the throws that handles if the arguments were too long or null. Instead of having these as two separate throws or fail instances I included them in the same one in an “or” statement within an “if” statement. As mentioned, before I followed proper naming conventions and I also kept the indentations all proper. Most of the code in the simple class files kept the same structures and the code in the service files also kept the same structure for the most part. This is very important because once you read one file and understand it well you can read others with more ease as it follows almost identical structure. I also condensed my code as much as I could to make it smaller like with the setters in both files.

**What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

Since all of the tests that I created were based on the specifications that were provided for the program, we would consider my following techniques to fit into the category of specification based testing or black box testing. One technique that I tried to implement throughout my milestones and had a lot of trouble with was equivalence partitioning. This is basically where we try to limit the number of tests being written which helps decrease the time it takes to write tests and chunks similar tests into the same test cases. This is great for saving on time and for trimming down the size of the test files. Additionally, I did my best to implement exhaustive testing to help cover at least 80% of the code. I did this because I know that the code sections were not too complex, and it would not take me too long to be able to write tests that would cover everything. With more complex programs this could be more difficult and even impossible but with this program’s set of specifications it made sense. In a way I did also use boundary value analysis as my tests did have boundaries however that implementation was very limited.

**What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.**

Some other types of software testing techniques that I did not use was decision table testing and state transition testing. A decision table follows the idea that functions respond to several inputs in order to proceed with that functions task. For example, if you are entering in your billing information into a system and the save button does not become clickable until the last valid piece of information is entered. Another example could be when you are creating an online account and all your account details need to be completed before you can create the profile. It follows the idea that for a specific task to be completed so many boxes must be checked. State transition testing is often used in tandem with decision table testing and is most often used when the testing of a sequence of tasks needs to be done and when the program needs to be tested for a specific set of input values. Now technically I did implement tests that accepted certain input values within a range but that fell under boundary value testing in my case.

**For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Using equivalence partitioning is a testing technique that I believe can be used in every piece of testing code. That is because I could not think of a way in which decreasing the test size and condensing it to be more efficient would be a bad idea. Making a partition of test cases based on the requirements seems like it can be done with almost any set of test cases, given that there are enough of them. Boundary value analysis is interesting because the amount of implementation really depends on how much equivalence partitioning takes place as the two testing techniques are closely related. By testing at the boundaries of values within partitions, this technique would be well to compliment most equivalence testing situations. Some practical applications could be a grading system for a school where certain grades are different boundaries. As described above, decision table testing is very well suited for applications where all the boxes need to be checked in order to proceed. We see this mostly with websites where entering in personal information to create accounts or for billing is needed. It would be bad if a website let you checkout an item before you entered in your credit card number so the practical applications for this type of technique are huge and very useful in online applications.

**Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

While working on this project and acting as a software tester I had to change my mindset while building the program and while testing it. When building a program, I would typically break down the situation I had into smaller steps that I could start coding out and making something that worked. Now, I followed this same method for the first parts of the project as developing a working program was important, yet what through me off and gave me a hard time was when I didn’t exactly have an output for the program. The project takes two files which create and set the stage for an object and then allow for manipulation of those objects. Getting this initial code didn’t take much thinking in the way of a tester but when I started to write my tests my entire mindset changed. I had to start thinking less in a creative manner and more in a destructive manner. In other words, I had to think about how I could break my code that would violate the requirements of the project. This was challenging as I have never done this before and was also very time consuming. For example, I knew I needed to keep the number of characters for certain strings down to a certain minimum but applying the test for that took me a while at first. This is where I started to appreciate the complexity and relationship of the code I wrote and the testing I was doing. Working in smaller chunks really helped me build tests faster and more efficiently.

**Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

When reviewing my code, of course I thought my code was great and worked well, however, to eliminate bias I would code the assignment early in the week and start to build the tests later in the week. This would give me a fresh set of eyes to look at the code as if I were someone new to the process. This helped me find errors in my code and ultimately helped me build better tests. I can imagine, where there is a time crunch, that most programmers don’t have time to take a break from code before testing and this could be a huge problem. Programmers often treat their code in a favored way and in many cases, once the code works, the programmer is very happy that it works and depending on time, not much is done in the way of optimization. With that in mind, having someone else handle most of the testing is a smart move so that you can avoid bias and uncover problems that a new set of eyes would be able to find that would be more critical.

**Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

The importance of being disciplined in the quality of the code that I will be writing is something that rests in the back of my mind quite often. Whenever I get hired by a company or organization to start using my coding skills, I want to do the best that I can to produce something that is clean and makes sense while following all ethical avenues. Cutting corners can result in errors and errors essentially cost the company money and time. If I were to start costing a company a lot of time and money than I could potentially lose my job or receive disciplinary action. I want to be proud of my work and happy knowing that what I was able to produce may not be perfection but will be the best I could do in the allotted time. I can void technical debt by following the requirements set out to me by the team, remembering to complete unit tests in an effective and timely manner, and by refactoring my code to make sure that it is following requirements, following company policy, and providing an example of effective testing techniques that will reduce time spent on errors.