Cory Doak

Southern New Hampshire University

CS320: System Test Automation & QA

June 18, 2023

Project 2 Summary and Reflection Report

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The software requirements of the assignment included the objects within the class had to be a specific length and could not be NULL. In order to ensure that all three features, the contact class, task service, and appointment service classes, met the requirements we had to specify in the code that these requirements were met or else they would not pass. For example, in the Contact Class, we needed the contact’s first and last name each could not be more than 10 characters. The phone number had to be exactly 10 characters, and the address could not be longer than 30 characters. In the code, this was established in lines 13-27 of ContactClass.java.

*if(firstName == null || firstName.length() > 10) {*

*throw new IllegalArgumentException("Invalid First Name Input");*

*}*

*if(lastName == null || lastName.length() > 10) {*

*throw new IllegalArgumentException("Invalid Last Name Input");*

*}*

*if(phone == null || phone.length() != 10) {*

*throw new IllegalArgumentException("Invalid Phone input");*

*}*

*if(address == null || address.length() > 30) {*

*throw new IllegalArgumentException("Invalid Address Input");*

*}*

*if(id == null || id.length() > 10) {*

*throw new IllegalArgumentException("Invalid ID Input");*

*}*

You can see this again in the Task.java and Appointment.java classes as we initially set the boundaries of the features. If the boundaries were not met, then we threw an argument exception that read “Invalid Input” based on the type of input that was required.

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

Based on the coverage percentage of the Junit testing, the tests were effective. We tested inputs that would give us a false return, as well as a true return to be sure that these tests were functioning properly. The goal was 80%, and this was surpassed as majority were testing at 100%.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

Ensuring the code was technically sound, we went ahead and practiced different methods of tests. We tried by throwing illegal arguments and giving the inputs to be expected. If these inputs did not match, then the test would fail. As mentioned, we would test both sides of the expected function to make sure it would test as passing and failing. In TaskTest.java, we tested the adding a task function, and here in lines 13-15 we tested with a specific input.

*taskService ts = new taskService();*

*Task testTask1 = new Task("123456789", "Test Task", "This is a test to test the test task of this code.");*

*assertEquals(true, ts.addTask(testTask1));*

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

Making sure the code was efficient, we avoided any complex configuration. We kept test cases simple and to the point. For example, in AppointmentServiceTest.java, lines 20-23 we tested that the input of an appointment would test out properly. In doing so we provided a wrong Appointment ID input, in which should throw the illegal argument exception provided.

public void testAddAppt() {

Assertions.assertThrows(IllegalArgumentException.class, () ->{

new Appointment("1234567890.", new Date(2099, Calendar.JANUARY, 1), "This is a proper description.");

}

);

}

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

In the Appointment project more specifically, numerous amounts of tests were run to ensure that all the input requirements given by the company were met. Since the input requirements were given to us in the program specifications, this would be a black-box testing technique. We also utilized boundary value analysis where we set the boundaries of the requirements and were to make sure we stayed within the confines of these boundaries or fail otherwise.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

One technique we did not utilize within this project are decision tables. Decision tables are tables of rules, inputs, and outcomes that are used to outline the program specifications, and they can be used to help drive the design and creation of unit tests to ensure that specifications are met (Hambling et al., 2019).  In addition to these, structure-based testing techniques, or white-box techniques, we did not use within the project. A flow chart would be an example of a structure-based technique, in which we did not use for this assignment because these are more so visual aids to help the developers.

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

Each of the mentioned techniques have practical usage and implications. Black-box techniques usually used when there are changes in specific components of the feature, and the desired test is to only verify the changes that were made. Flow charts are for the developers to have a visual aid to their code development. Decision tables are used to help outline the program specifications, in which then could be used to help create unit tests. This would be more practical for a more complex testing and requirement management.

* 1. Mindset
     1. 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.

Acting as a software tester, I was able to practice caution in reading through the requirements and executions of the code. After, I could understand what my tests were to be testing and how they should be implemented. It is important appreciate the complexity and interrelationships of the code being tested because we had to make sure the boundaries from the classes were also being followed in the testing stages as well. The objects created all remained connected throughout the project as they were able to be called from one class to another. For example, we created the ContactClass.java objects contactID, firstName, lastName, phone, and address and we were able to bring them into play in ContactService.java when adding a new contact, updating the first or last name, or phone or address based on the contact ID. Then we were able to implement the tests where we could bring these objects in again and test their abilities to function properly.

* + 1. 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.

Attempting to limit bias would be easier if I was not testing my own code. Since I am testing my own code, I was very lenient on myself and showed grace on errors and tests. I believe that if developers were to test their own code, they may present simple tests to show the passing coverage percentages, but future development would pose issues as these tests may not be as complex as necessary. I was able to fix any code of my own as I placed tests and failed. Whereas tests failing in someone else’s builds would just be sent back to the developer to correct their own errors.

* + 1. 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.

Staying disciplined and committed to quality plays a huge role in software development because without quality assurance, the whole project could a be a disaster. Looking at development failures (such as the couple we read about in the resources this week) is a prime example to why testing is important in software development. Software is incorporated into everything around us these days, so quality is the most important aspect of development. Cutting corners leads to errors, issues and problems in the future and it is important to test as you go rather than testing is all at the end. To avoid technical debt, I would practice testing as I go to fix errors as I find them and utilizing tools and techniques such as decision tables and flow charts to help organize the project prior to the initial development. Accruing technical debt only causes already existing problems to grow and get worse over time, which means it is more expensive repair. It is best to fix the errors sooner rather than later.

**References:**

Doak, Cory. Project 1. June, 2023. Course CS320: Software Test Automation and QA, Southern New Hampshire University.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing : An istqb-bcs certified tester foundation guide - 4th edition. BCS Learning & Development Limited.