7-2 Project Two

Luis Valdez

Southern New Hampshire University  
CS-320-H7026 Software Test Automation& QA

Professor Ramsey Kraya

December 10, 2023

**Summary**

The best example of my approach to software requirements alignment was, in the contact.java file. I had a feature that made the requirement of limiting names to a maximum of 10 characters. I believe this was very important because, in the project requirements, it was noted that it could not exceed 10 characters. I made sure to include comments across my code to deliver high-end work for the customer such as “Validates the contactId against null and length requirements.” This was crucial for my JUnit tests also. I had to ensure that this tested my constraint. The goal of obtaining at least 80% coverage in JUnit tests is another measure of alignment was also another approach I took to provide specific evidence. Overall, these were some ways I met the requirements through effective testing practices.

The example of my code that I used to ensure my approach aligned with the software requirements:

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

**throw new IllegalArgumentException("Invalid ID, Try Again.");”**

I had to make sure that I covered the score of 80% or above was my goal because it meant that it was a completed test. Also, if I had a high percentage this would mean that a large section of the codebase was tested, so that worked well for me. I was able to figure out defects and mistakes through testing in Eclipse/Visual Studio. I also like to use XCODE for IOS (XCTest). The debugging feature is very helpful for me when figuring out the problems that arise.

To ensure that my code was technically sound, I ensured through fundamental coding practices. I used the setter’s method below to confirm that the first name was 10 characters or less. In my contact class. This not only fulfilled the requirement but also enhanced my code.

**// Setters for each field**

**public void setFirstName(String firstName) {**

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

**throw new IllegalArgumentException("Invalid first name: must be 10 characters or less and not null.");**

**}**

**this.firstName = firstName;**

**}**

**Reflection**

For efficiency, I employed my strategy to use my code's illegal arguments. It is efficient because my code could be worked with multiple settings to test other wrong data. The reuse of this testing portion reduces the steps and makes it easier across multiple test cases. Lastly, these steps make sure that tests are not only effective but that they save time. Which means that it is an efficient testing process.

**public void setLastName(String lastName) {**

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

**throw new IllegalArgumentException("Invalid last name: must be 10 characters or less and not null.");**

**}**

**this.lastName = lastName;**

**}**

I had several software however, the primary technique that I used was JUnit testing. This was done in Eclipse to mainly focus on validating each code component. Another technique that I used that I have not touched on was static testing. I was able to learn that this type of testing is a to identify errors early in the development process. Static testing is a great technique for complex projects because you can find errors early on as opposed to later in the SDLC. Lastly, static testing allows developers to examine code without running it in a simulated environment.

I have discussed several techniques but to reiterate, Junit tests are ideal for testing individual components of an application. Typically, these tests are done isolated as opposed to testing the whole code. I use the @test in my code to display. I would consider this method very helpful because I like to follow the incremental coding strategy to make sure I am on task and each line of code is efficient. The implications are that they help identify bugs at an early stage, which leads to fixing bugs incrementally.

**Mindset**

Mindset and Bias in Software Testing Throughout the project, I maintained a cautious approach because in my experience this was the most complex task I have ever been assigned. The final product is a tough task to accomplish. I appreciate the complexity and interrelationships of the code because while I was testing, I also worked on my discipline. To try to limit bias in my code review I had to remain focused for long periods. I believe yes, I would be concerned if I was responsible for testing my code. I had difficulty deciding on the design decision, so I overcame it by researching and testing. I have witnessed the type of pressure there can be whenever you need to make sure that everything goes right and that everything has its right purpose. Cutting corners when it comes to writing or testing code only hurts you in the long run and will have to end up getting fixed in the end. I will continue to use static testing in my work in the future. I am sure that if my end goal is to prioritize quality over speed, it puts me in a better position to make sure that the program succeeds. I have always had a passion for learning and attributing my growth to programs and systems.

**References**

Devi K Kalpalatha. (2023): What is Static Testing? <https://www.browserstack.com/guide/static-testing>

Vivek Mannotra (2023) What is iOS Unit Testing? (Tutorial with XCode & Swift): <https://www.browserstack.com/guide/ios-unit-testing-tutorial>

Stefan Bechtold Sam Brannen Johannes Link Matthias Merdes Marc Philipp Juliette de Rancourt Christian Stein Version 5.10.1 (N/A) JUnit 5: [JUnit 5 User Guide](https://junit.org/junit5/docs/current/user-guide/) & [JUnit 5](https://junit.org/junit5/)  
  
Apple, Testing your apps in XCode Apple Developer Docs: <https://developer.apple.com/documentation/xcode/testing-your-apps-in-xcode>

SNHU Media (2020) JUnit Tutorial: <https://www.youtube.com/watch?v=KbXhK9HUng4>