Ryan Blegen

CS-320 Software Test Automation

Project Two

4/21/2024

Unit testing is a crucial practice in software development that focuses on verifying the correctness of individual components in isolation. For this project there were several requirements the methods had to meet. To meet the requirements for this project I utilized JUnit testing to check the methods. For example, in the Contact class there were requirements that it have a unique ID that cannot be changed, cannot be longer than 10 characters and cannot be null. To accomplish this, in the Contact class we stipulate that if the contactId is Null or the length is greater than 10 it will throw an illegal argument exception. The following snippet from my code demonstrates this:



My JUnit tests were effective based on their coverage percentages. For all classes the coverage percentage was 100%. Methods and line coverage varied between 80-100% coverage. We were required to meet a baseline of 80% coverage. So based on these results I am confident in the overall quality of my tests. In addition, all the test classes passed, meaning they met the requirements being tested.

To ensure that my code was technically sounds I followed several secure coding practices. I manually reviewed my code for errors, tested often, and utilized features in IntelliJ like syntax highlighting to spot errors. To ensure readability I kept my code simple and easy to understand, making sure to address only what a specific class was required to do and utilized the same code for classes that had similar requirements. I adhered to java best practices of encapsulation by keeping class members private, for example in my Contact class the members are private:

A screenshot of a computer code

Description automatically generated

In addition, I made use of the final modifier to ensure that the contactID could not be changed from its initial value.

To ensure that my code was efficient I focused on writing simple code that only addressed the given requirements. Using overly complex code has a higher chance of introducing errors and requires more time to debug. The efficiency of my code is demonstrated in my unit testing as it only took 59 milliseconds to run all tests. Writing clean and readable code is essential for ensuring that code is efficient. For my test classes used simple labeled code, only changing the parameter being tested. For example, this class for testing if the ID was too long:

A computer code with text

Description automatically generated with medium confidence

Labeling your code is essential for readability and documenting everything purpose.

The software testing techniques I employed for this project mainly consisted of JUint testing. JUnit is a popular open-source testing framework for Java that allows developers to write reliable and efficient unit tests. It ensures that the program logic works as expected by creating and executing test cases. While JUnit testing is effective for testing the functionality of individual components there are several other software testing techniques that may be better suited to use depending on the programming language and use case. For example, if the developer was testing code where they do not know the internal design they may want to use Black Box testing. Black Box testing examines the internal structure of the software, including code, logic, and control flow. It focuses on ensuring that all code paths are tested. Choosing the right testing method for your specific needs is critical to effectively testing your code.

I tried to keep an open mind when developing this project. I have limited experience with Java, and it has been some time since I wrote any code in general. I think it is important to appreciate the complexity of interrelationships of the code you are testing because it can help you better understand all the components and how they work together. The better you understand your code the easier it is to fix and streamline. For example, understanding the relationship between the Contact class and ContactServices is important to understand how they feed information between each other and into the test cases.

Recognizing bias is important as a software developer. If someone is testing their own code, they may not give testing the full attention to detail that it requires. They may convince themselves they wrote it correctly or in the most efficient way initially when there is a better way of doing it. Through the development of this software, I learned a lot about java and better ways of writing code. This led to a lot of rewriting and reworking code that I had already written. For example, in my initial attempt developing my ContactServices class I used an overly complex method for storing my contact information. While writing the code for this project I later decided to use a simpler HashMap to store the contacts. This cut down on my code, making it simpler and easier to read.

These projects were difficult for me. My java experience is very limited and requires a lot of reading and research to determine the best way of meeting the requirements. Good discipline is important for developers to stay up to date and knowledgeable. Initially I used a lot of techniques that were not necessary the best or most practical way of coding. I experienced a lot of technical debt by the end of this project. This resulted in a lot of time spent going back and rework/rewriting my code. I think the best way to avoid this is to do your research, reach out to others with more knowledge and get their advice and stay knowledgeable.

Work Cited

Stefan Bechtold, S. B. (n.d.). JUnit 5 user guide. https://junit.org/junit5/docs/current/user-guide/

*Software testing techniques: Explained with examples*. BrowserStack. (2023, May 25). https://www.browserstack.com/guide/software-testing-techniques

GeeksforGeeks. (2024, March 14). *Black box testing - software engineering*. https://www.geeksforgeeks.org/software-engineering-black-box-testing/