Summary

Testing for this project proved to be rather challenging for me. In my initial coding of the various files I used a different approach to my testing. Initially I used more of a one-to-one approach and found that it helped me to organize my thoughts better, but overall did not flow well with the code nor was it easily adaptable once I got feedback on my code. For Project One I wanted to ensure that I addressed the given feedback which required reworking and lots of research on other approaches to Junit testing. I found initially that I was allowing for a null phone number which should not have been the case as the requirements were 10 digits or greater. Therefore, this time around I implemented the ContactTest assertion of

“@DisplayName("The phone number object shall have a required phone number string that cannot be longer than 10 characters.")

void getPhoneNumber() {

Contact contact = new Contact("12345", "Sar", "Harrison", "7607101402", "123 Fake St");

contact.setPhoneNumber("7607101402");

assertEquals(contact.getPhoneNumber(),"7607101402");”

Through this method, I was able to ensure that the 10 digits was a requirement and could not be null. I felt the overall quality of my testing improved and felt less clunky compared to the first time around. While my coverage percentage was higher during the first round, I noted that this time while the coverage was lower, it fell within the desired range for Junit testing and was therefore more efficient.

I also made an attempt at making my code more efficient and technically sounds using techniques such as in Contact.java where I identified my variable and then coded for my getter and setter during which I assigned values as opposed to separate sections for each. This allowed for greater organization on my part when creating tests:

public String getContactID() {

return contactID;

}

private void setContactID(String contactID) {

validateContactID(contactID);

this.contactID = contactID;

}

I tend to struggle a bit with coding so I made sure to watch many tutorials and to test frequently to avoid too much backtracking and errors in the long run which I feel helped in my revisions process overall.

Reflection

Like the first time around I chose to employ Black Box and White Box categories of techniques as these make the most logical sense to me and are structure based so I can apply similar logic to them that I would my code itself. Black box testing and white box testing are two complementary approaches to ensuring the quality and reliability of software during the development process. Black box testing focuses on evaluating the software's functionality from an end-user perspective, without knowledge of its internal code structure. This method helps identify issues from a user's point of view, enhancing user experience and ensuring that the software meets its intended requirements. In contrast, white box testing examines the internal code, logic, and structure to uncover issues like security vulnerabilities and coding errors. Both techniques combined provide a more comprehensive quality assurance process, covering both user satisfaction and code thoroughness. Seeing the code from both aspects between the helped me to organize and ensure I am testing the correct components from a logic standpoint. Being able to break down my testing into sections similar to my code itself also ensured that I did not do too broad of a test. In my previous round of coding, I felt that my testing was not efficient as my coverage percentage was too high and therefore, I was testing more than necessary. This time around I noted a lower percentage that still fell within the desired range.

Similar to last time, I was not able to utilize experience-based techniques. Experienced-based software testing, often referred to as heuristic testing, is an approach where testing is guided by the tester's knowledge, intuition, and expertise rather than following predefined test scripts or formal test plans. Testers draw upon their experience and domain knowledge to identify potential issues and test scenarios that may not be apparent through structured testing methods. This approach can be particularly valuable for exploratory testing, ad-hoc testing, and uncovering unique, unanticipated defects, making it a versatile and dynamic way to ensure software quality. Experienced-based testing allows for flexibility and adaptability in response to evolving project requirements and changing software features. While I did work with this project before, I ended up reworking my code for personal purposes and found that I needed to use more structured testing methods as I am still unfamiliar and unsure.

Over the course of this project, I felt that I adopted the mindset of constantly learning. I noted that I employed quite a bit of caution and was very unsure. That being said, however, I feel that my catious approach motivated me to do more research than I otherwise would have in order to ensure I was creating quality code. I found that tutorials and open-source platforms helped to teach me the nuances of both coding and testing over the course of this project and helped me to develop a deeper understanding. It was amazing to see the correlation and interrelationships of the code I was testing as it helped me to realize the importance of proper code syntax, organization and general industry best practices. The smallest of mistakes in one area caused huge errors in other areas that seemed way worse than they actually were. The interrelationship also helped me to understand the importance of another set of eyes testing vs. coding as there is an inherent bias employed when doing both. I was much more willing to go back and tweak my code as I went and, due to my lack of confidence in my coding abilities, found myself over-testing the first time around as I was sure that I made more errors than I actually did.

This experience has taught me the importance of discipline in a software engineering role. I determined that, in order to ensure my own success, I needed to develop my own set of standards and become more familiar with industry best practices as the code that I write will be utilized my other programmers or even myself later on and if the quality of the code is poor it causes more issues than necessary. Cutting corners does the developer a disservice as it builds poor habits and can impact all aspects of all of their projects which not only impacts them the project team as a whole as well as the client. Writing quality code helps to reduce the waste of developer and company resources by producing a reliable product upfront and helping to avoid backtracking later on or potentially another aspect for the program failing due to unnoticed errors in code that could have otherwise been avoided. For example, too little testing may lead to code that breaks or fails at a later stage in development or even after deployment and can aither cause a disruption in the SDLC or even a security risk/break for a client using the failed program. Overall, it is better to put the time in upfront than to have to clean up a mess later on.

Sources:

Wall, P. (2023, August 14). *What is Black Box Testing | Techniques & Examples | Imperva*. Learning Center. <https://www.imperva.com/learn/application-security/black-box-testing/>

Michali. (2022, March 28). *What is White Box Testing?* Check Point Software. <https://www.checkpoint.com/cyber-hub/cyber-security/what-is-white-box-testing/>

Hamilton, D. (2023, August 18). *JUnit Tutorial With Examples: Setting Up, Writing, and Running Java Unit Tests*. Parasoft. <https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests/>