CS-320: 7-2 Project Two

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

In order to align the project to the software requirements, I made independent tests that focused on one function. Then, I tested for as many boundaries as possible. For example, contactIdConstructorTest tests that the contacttId passed to the constructor is equal to contact.getcontactId. The first name, last name, phone number, and address are not null, and the information is being read correctly. Testing these ensures there are no issues with the constructor. There were also tests to check that the first and last names were no longer than ten characters. The project set limits like this.

* 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?

With a 93.4% overall coverage rate, the JUnit tests were adequately spread throughout the entire program. ContactTest had the highest coverage overall. Each test has its own responsibility and is isolated to ensure one failed test does not affect another. Testing boundaries for the minimum, maximum, and unique values also helps with the JUnit’s effectiveness. Although coverage never hit 100%, the main processes were covered, and there was a consistent improvement every time.

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

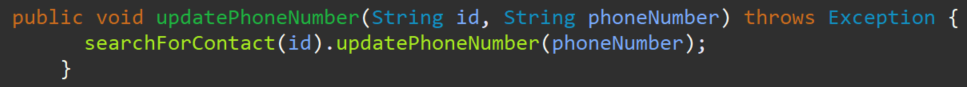
To ensure my code was technically sound, I made sure to keep my formatting consistent and followed coding standards. I also used clear variable names for easy identification. I tried to reduce redundancy by classes, functions, and methods to encapsulate standard functionalities. I was able to lay out the program visually beforehand to get a design and structure before building the project.

A computer screen shot of a code

Description automatically generated

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

I tried to keep my code as efficient as possible by avoiding any extra characters. Sometimes, I realized an easier way to implement something and would go back and rewrite the section. From there, it is easier to reuse the code for later use. Clearly labeled variables made it easy to change them as necessary for the next test.



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

Various testing techniques cater to different aspects of testing. The coverage tests used Structure-based testing extensively to analyze components and if-then statements. It is used to break down the tests into sections. Structure-based techniques include statement coverage, path coverage, and branch coverage. Unit testing was the primary test for this project. It focused on individual functions of code in isolation. Multiple tests were performed on the same principle to test different boundaries, so boundary testing was initiated. I ran tests against the requirements of the project for functional testing. Constant regression testing occurred as new code was implemented to check if the previous code still worked correctly. One could say that compatibility testing was performed for a Windows 11 device, as that is my operating system. I would periodically perform negative testing by inserting invalid inputs to see if an error would be returned.

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

There were no experience-based techniques performed in the project. Also, no performance testing was completed, as it was not a requirement. So, the software’s speed, responsiveness, and scalability were not checked. No security testing was performed. Vulnerabilities in the code could still be a factor. No alpha and beta testing was completed, so no real-world feedback was gathered. A/B testing was not performed, and I did not compare the newest version to earlier ones.

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

The techniques described above derive from code development's black-box and white-box sections. Black-box is used when there is a clear-cut functionality and is best used for outsourced testing. White-box is used when the desired outcome of a product is known. Experience-based techniques are used where specifications are either missing or inadequate and time constraints exist.

* 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.

As I have never done unit testing before, I mainly focused on experimental and growth. I tried to tinker with different settings to see what they did. I would employ caution when in a business environment to find the boundaries and limits of a program quickly and check those confines consistently. It is essential to appreciate the complexity and interrelationships of the code as it impacts the quality and performance of the final product. My first coverage was 37%, and I thought I was doing fine with my project. As I raised that percentage, I realized mistakes that could be encountered in everyday scenarios that could throw an exception or crash the program altogether.

* 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 code? Provide specific examples to illustrate your claims.

When making a design, it was hard to admit that my initial theories weren’t working and that I needed to go in another direction. I also forced myself to run a test after every function, regardless of whether I knew it would work or not. I fooled around with the tests and tried to make them fail to verify the test was indeed working. One example would be trying a string that was eleven characters with a ten-character limit requirement.

* 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.

Every time you cut corners, you risk making a mistake that could diminish your product's quality. This, in turn, could result in lower perceived quality and, ultimately, less business. Customers routinely gravitate towards reliability when it comes to programming. Having buggy code is frustrating and can result in productivity loss. It is essential to be disciplined in the commitment to quality as a software engineering professional because it helps uphold and “advance the integrity and reputation of the profession.” (Software Engineering Code, 2018).

References

Software Engineering Code. ACM Ethics. (2018, December 19). <https://ethics.acm.org/code-> ofethics/software-engineering-code/.