Cory Hurlbut

CS320

8-9-23

7-2 Project 2

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.
      2. 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?

Each feature had similar requirements and functions. I directly translated the requirements into tests then matched the functionality of the code to satisfy the tests. An example of this would be my test for successfully updating a contact’s first name.

Requirement – first name must be updateable, first name cannot be null, first name cannot be more than 10 characters.

Test –

@Test

Void testUpdateContactFirstName() {

assertTrue(contact.setFirstName(“12345”, “Jimmy”));

}

Code –

Public Boolean setFirstName(String firstName) {

If (firstName.length() > 10 || firstName == null) {

Return false;

}

This.contactFirstName = firstName;

}

Coverage percentage was over 80% which was the requirement. I noticed the test cases themselves brought the coverage down as it should have been closer to 95%. This is a good result for such a small program.

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

public boolean deleteAppointment(String id) {

for (int i = 0; i < appointments.size(); ++i) {

if (appointments.get(i).getId().equals(id)) {

appointments.remove(i);

return true;

}

}

return false;

}

Above is a great example of efficient and technically sound code as it is written to finish as soon as it accomplishes what it needs to. It also is concise and readable, making it maintainable.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

I have mostly used unit testing and functional testing to ensure specific functions perform as intended. This is the main use of writing code to test. JUnit is also meant to be used to test units of code. There are software that test code on a larger abstract scale but we did not use them. Each of my test cases test a specific function with positive and negative results. I also tested each part of the function using different possible parameters. An example is the appointment tests I wrote for each negative test per parameter.

I did not use acceptance testing, integration testing, regression testing, performance testing, or security testing. These are all ways to test different aspects of software. Acceptance testing is for testing if it meets the user’s satisfaction. Integration testing is to make sure parts of the code work together properly. Regression testing is to make sure new features do not break old functionality. Performance testing is to make sure the software is performant and does not have any issues in acceptance. Lastly, security testing is as it sounds, testing for security vulnerabilities or loopholes.

* 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.
     2. 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.
     3. 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.

One way I used caution was by adding a before each and after each function to the test cases. This set up service instances and cleared them to make sure that each test had the same start and would not cause unexpected errors.

I have always had to test my own code and I can see the potential problems with doing so. It can lead to oversights as your understanding of what your code is supposed to do can interfere with finding what it cannot do.

Cutting corners in coding can lead to many unexpected issues. The more important information a system holds the more disciplined programmers must be. If the most important info on your site is your favorite color, you don’t have to worry too much. But if the site holds national secrets or accounting info, you will want to quadruple check there are no possible ways that people can do things that you don’t intend them to do.