CS-320

Professor Gudivada

Cheyenne Nave

August 14, 2024

Project 2

**Summary**

* Describe your unit testing approach for each of the three features.
  + The three features for Project 1 consisted of a Contact feature, Task feature, and an Appointment feature. Each of these features contained different functions which caused them to behave similarly in some ways, and different in others. The Contact feature required Contact information that could be added, removed, searched, and updated. Because each contact had a unique ID that could not be changed, each of the functions had to work with the ID. To test this, I had to build my tests around the Contact ID and its use in the Contact Service.
  + With the Task feature, the requirements were that each task has a unique ID, a set number of characters for a name and a description, and each task could be added as a new task, deleted, or updated. Again, the ID was used to search for individual tasks, however my testing approach here changed to accommodate a longer description (50 characters) and I had to prove that it actually updated once the change was made.
  + The Appointment feature was like the unique ID, date, and description and could only be added or deleted, not updated. Again, the unique ID was used as an anchor for the testing for searches, deletion, and addition, however, it got a little tricky when it came to using the date variable. I had to back-date the entry date using a built-in calendar to make sure the tests would fail if the date was before the current date of the appointment setting. After a few failed attempts at coding the test, I finally figured it out by using the dot operator (i.e. Calendar.january).
* Describe your experience writing the JUnit tests.
  + Before this class, I had zero experience writing JUnit tests. I really had to hit the ground running to get a basic understanding of what JUnit testing consists of, how it applies, and why it matters. I ended up importing some functions from the JUnit library, specifically *assertNotNull,* and *fail.* For example, line 19 in my taskTest class reads: “*fail*("Task ID is too long.");”. *Fail* from JUnit acts similarly to a Boolean value in that if the test definition (in this case: “if(task.getTaskId().length() > 10)) was true, then the test would flag and fail it.
  + To ensure my code was efficient, I used *DisplayName* from the JUnit library as well. *DisplayName* allows a developer to give a tag to a test so as the developer is running tests, they can read the displayed tag and associate it with which task failed. In taskTest, I used “@DisplayName("Task ID cannot exceed 10 characters")”. This is the displayed tag for my test that checks the length of the unique ID. If the test fails and flags in the JUnit panel, this specific display name will show in the console.

**Reflection**

* What were the software testing techniques that you employed in this project?
  + I mainly used Functional Testing and Unit Testing. Functional testing focuses on the requirements of a project and if they are met. Unit testing makes sure that each individual component of the project works as it should. Each class in Project One (Contact, Task, Appointment) needed to be tested functionally to make sure that each met all the requirements of the rubric, and they also had to undergo unit-tests to make sure each class worked as it should individually.
* What are the other software testing techniques that you did not use for this project?
  + For this project, I did not need to use other types of testing such as System Testing or Performance Testing. System testing involves testing the entire system while Performance Testing checks responsiveness, speed, and stability of the program. These were not needed based on the requirements given.
* For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.
  + Functional testing is vitally important because it makes sure the project meets the client's needs. This could be anything from a simple calculator app to a high-level iOS developed project that needs to be checked before being deployed to the customer base. If there are errors in said application, Unit testing can come in and check each class of the project to see if it’s working the way it should. This whittles down the process of finding the error. In this situation, Performance testing and System testing would be necessary because the developers would need to see what load is going to be put onto the customer’s hardware and how the application reacts. User experience, quality, and usability are all affected if the tests are poor.
* 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?
  + I caught myself trying to code “defensively” when building my regular classes. I didn’t want to deal with excess variables or multi-level loops that could end up dense. I had to build the classes with the tests in mind because I had to think about how to use each object and how to test. For example, when creating my Contact class, I wrote:
    - if(firstName == null || firstName.length()>10) {
      * throw new IllegalArgumentException("Name is too long");}
  + The following is the test for the code:
    - @Test
    - @DisplayName("First Name has too many characters.")
      * void testFirstNameTooLong() {
      * Contact contact = new Contact("id", "Luke Skywalker", "lastName", "phoneNumber", "address");
        + if(contact.getFirstName().length()>10) {
        + fail("First Name has too many characters.");
  + In this example, simple variable names are used, and many actions are simplified with the dot operator. This allowed for easily readable code and better understanding of what is happening.
  + Also, I had to understand the complexity and interrelationships of my code with the other classes because if I didn’t, then my tests may have not succeeded. One of the requirements for the Contact class was that each contact had a unique ID. I had to ensure that the tests worked, and unique IDs were built for each contact because the Task and Appointment classes worked based on the Contact ID. If I didn’t test the ID in the contact class, my Task and Appointment classes could have been thrown or displayed improper test results.
* 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?
  + It was incredibly hard to limit bias overall. Through Project One, I learned that bias would most definitely be a concern if I were responsible for testing my own code. I especially had a hard time with Confirmation Bias which ultimately is a type of Bias that explains testing using data that proves a test rather than disproves it. In doing so, failures can go unrevealed because the biased testing proves the program’s functionality.
* 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?
  + It is so incredibly important not to cut corners when writing or testing code because every little character matters. If a semicolon is missing, the program will not run. If a variable name does not match exactly, an error will be thrown. If minute discrepancies such as these prohibit a program from running, then major errors will do the same – at a much higher technical debt - so it is important to check and test for failures or bugs in your program. To avoid this issue, it is important to understand that testing can start at the very beginning of the Software Development Lifecycle. This will dramatically cut technical debt and potentially save the monetary budgets of clients as well.

**Cites**

* *CSDL | IEEE Computer Society.* (2023, December 10). https://www.computer.org/csdl/journal/ts/2023/12/10313064/1RUgF7OmNPi