Brooklen Ashleigh

Professor Wilson

CS320

August 12th, 2023

Project Two

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

I intentionally built my Junit tests so that they would test for the functionality outlined in the program’s functional requirements. For example, in the Task class, the ID of the task must be unchangeable and with a max of 10 digits. The name attribute of the task class cannot be longer than 10 characters. The description attributed cannot be longer than 50 characters. All three attributes cannot be null. To test for this important functionality, I created Junit tests that try to instantiate a Task object that violates each of these parameters individually. We assert that, for example, trying to instantiate a Task with a name that is over 20 characters long will throw an exception, and when it does, that Junit test passes.

A screen shot of a computer code

Description automatically generated

**Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

Both my Task and Contact classes are at 100% test coverage. My ContactService and TaskService coverage percentages are at 96% and 97%. I intended to complete 100% test coverage, but I was not able to figure out how to satisfy two if statement branches. For example:

if(Tasks.get(i).getID().equals(ID))

or

if(findTask(task.getID()) != null)

Both statements have two branches, one of which is satisfied. I tried different methods of covering the second branch of functionality but could not complete it. I don’t think this lapse in coverage introduces any risk to the application, and do not believe it affects the quality of the code I’ve written. Overall, I have covered the possible failure points of each class to the best of my ability, especially with the new TaskService tests. I was able to use multiple different assertion types and created variables for my test values so that they would be easy to parse for other testers, or for myself testing again later in the development cycle.

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

In my test creation, I attempted to cover all failure cases as defined in the client’s requirements. For example, the name attribute for the Task class cannot be longer than 20 characters and cannot be null. To enforce this rule throughout the program, I created several tests to ensure that the value was not longer than 20 characters and wasn’t null during both instantiation, but just as importantly in setting and updating.

In the Task constructor, the following throws an exception if someone tries to instantiate a Task with a name field that’s too long:

**A screen shot of a computer code

Description automatically generated**

Then, if someone tries to set the name with a length greater than 20 or null, the setter throws the following exception:

**A screen shot of a computer code

Description automatically generated**

In the Task Tests, I assert that if someone tries to instantiate a Task with an invalid name, it will throw those exceptions. These tests verify that functionality:

A screen shot of a computer code

Description automatically generated A screen shot of a computer program

Description automatically generated

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

To ensure that my code was efficient, I made sure through exception throwing that objects are not instantiated or added to the object-holding list array if they have invalid properties. In a worst-case scenario, without proper validation, objects could be instantiated with duplicate IDs or improper attributes. While those improper objects could be sought out and cleaned up by another method, it is much more efficient and less costly to a system to stop those objects from being instantiated in the first place.

The following test ensures that an object with a duplicate ID cannot be created:

A screen shot of a computer code

Description automatically generated

The following code ensures that objects with improper parameters (in this case a too-long description) cannot be instantiated:

A screen shot of a computer code

Description automatically generated

**What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

For the Project One milestones, I utilized Unit testing to validate my code. Unit testing focuses on the smallest building blocks that code can be broken down into—classes and methods. I used JUnit testing to accomplish the goal of validating the Contact, Task, and Appointment classes and their methods.

The first part of testing is setting up code to handle errors and check for invalid data. Though the code itself is simple, data validation handled by an object’s constructor class is a powerful way to ensure that objects are not instantiated with invalid parameters. In the constructor of the Contact, Task, and Appointment classes, specific if statements are used to ensure that the data passed as parameters to instantiate the objects fit specific criteria, usually a maximum character length and a requirement that they are not null. If any of the conditions checked for are true, an exception would be thrown and the object would not be created. Those barriers are then checked with unit testing to ensure that an object can’t be instantiated with data that could cause a defect later on. Because objects can be modified, I also added specific rules to my setter methods so that the object variables could not be *changed* to something invalid after instantiation. I think it would also make sense to set up a class without setters where a system would create an entirely new object with copied parameters when a modification needed to be made to avoid redundant or potentially divergent parameter checking.

Altogether, these unit tests ensure that data is modified and created exactly how we expect it to on the most granular level.

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

On the developer level, unit testing is done alongside integration testing. In these three milestones, no integration testing was done to ensure that the three different classes and services will work together as a part of a larger system. For example, in the case of the Appointment class, I as a developer did not know what method the overall system would be using to pass the requested appointment date into the class so the date could be validated and a new Appointment object created. Doing unit testing would likely test that the system could use the classes I created to create appointments, tasks, and contacts. Beyond unit and integration testing, there is also System and Acceptance testing, which test that a program works with all components in place as an almost final product. That testing would be done with the inclusion of any potential databases, external services, UI, and driver interfaces.

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

Though the scale of a project might determine how much time and resources would be needed to accomplish all levels of testing, I can’t think of a scenario where Integration, System, and Acceptance testing should not or would not be completed in some form or another. Even the simplest of applications should be tested thoroughly to ensure there are no egregious errors in the code, some of which may only appear in very specific circumstances. Integration testing is needed to ensure all classes and parts of the program can work together. System testing is needed to ensure all components like databases are properly integrated and running. Acceptance testing must be done to ensure good security and that the program can run at scale and with good reliability.

**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.**

Testers must be extremely critical and detail-oriented in their approach to software testing. They must be able to understand exactly what is happening within a program, but also exactly what the requirements of the code are so that they can ensure that it will only function as expected. The more complex a system is, the more important this is, as data that comes from other systems or user input must conform to what is expected from it. In working on the tests for this project, I had to think of every possible way that the requirements for the project needed to be maintained and adhered to. My first mistake when I started writing my tests was in not properly testing my setter methods. The Appointment, Contact, and Tasks objects needed to be mutable, but I forgot to write tests that checked if the object could be *changed* to data that contradicted those requirements, where I had only focused on invalid instantiation. If I had tested properly, I would have seen very quickly that they could be changed to invalid data. It was an obvious mistake in hindsight, but I corrected it quickly.

**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.**

I limited bias in my review of my own code by not allowing myself to overlook any mistakes or perceived weaknesses in the code that I was writing. As a developer with strict deadlines and customers to please, I can see how it could be easy to say “this is good enough” and ship code that has a weakness, even if that weakness is unlikely to cause problems. In my case, writing J-Unit tests for the first time, I encountered points where I wasn’t sure how to test the functionality of an aspect of the code. Instead of trying to solve those problems, I could have passed them off as good enough and given up trying to figure out how to test it, or decided that my code coverage was hitting diminishing returns. However, I put in the extra work to solve and test for those aspects anyway. Once the test is written, it will be there through the development of the project and continue to ensure valid data. The cost of taking the time to write the proper test is insignificant compared to what may happen if you don’t.

**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.**

Security and consistency are two of the most important qualities that written code can have. It doesn’t matter what groundbreaking features a product may have, if they work inconsistently, continually give errors, or corrupt information, they are functionally useless, and a client will not want to use them. A reputation for creating bad programs can stay with a company forever, and losing customer trust can be irrevocable. The best way to ensure a good, quality product is to start testing it as early as possible. Doing a thorough job from the base level up will ensure that no time is wasted during the development process, that the project is easier to develop, and then easier to maintain. Organization from the outset is pivotal to ensure this. If an organization is not organized and intentional about its development process, it can incur technical debt, which is “the cost of additional rework caused by choosing the quickest solution rather than the most effective solution” (Team Asana, 2023). Intentional and unintentional debt can accrue when the time it takes to validate and properly test isn’t seen as necessary. Instead, that time and cost is pushed down the road to when the effort will have to be put into fixing something that already might be very heavily developed, which is harder and potentially far more costly.

Now that I have experience writing J-Unit tests, I have a better understanding of what I need to accomplish from the outset to make writing tests for my code easier. If self-testing, I plan to write my tests as I work, which will make it easier to ensure that I cover all aspects of my code and don’t miss anything in more complex projects. Though this can make refactoring tricky, tests are not terribly verbose, so I don’t think it will add wasted effort, but will instead ensure that I can write good, accurate tests, and make sure that I am working with a sound foundation as I develop. As stated above, the cost of taking the time to write the proper test is insignificant compared to what may happen if you don’t.

Resources

Team Asana. (2023, January 26). *What is technical debt & how to pay it off*. Asana. https://asana.com/resources/technical-debt