Cody MacLeod

SNHU 320

Project Two

For our most recent project we took on the role of a software engineer at Grand Strand Systems and created the services for the contact, task, and appointment services of a mobile application. One of the most important parts of this project was making sure the client’s requirements were understood and designing around that. Take the implementation of tasks and task services as an example. For tasks we had three listed requirements for a unique ID field, a name field, and a description field. Each field was listed with their own requirements, such as the name needing to be filled in, so it is not null, and should be 20 characters or less. Following these requirements, we could then set up conditional statements to ensure that if the input was greater than 20 characters or null that it would not be accepted.

To test that we met these requirements we utilized JUnit tests. Creating JUnit tests for every listed requirement would help us be sure that we have successfully implemented these needs. To best prove we met all these needs, we can look at the coverage percentage of our JUnit tests. Across all the tests, the lowest scoring one was the test for the appointment services which scored a resounding 86.7%, most of the others well above that. With such a high percentage of coverage, we know that we’ve tested each aspect of the code to ensure their meeting requirements.

As an example of individual JUnit tests, we’ll once again look at requirements for the task functionality. One of the requirements for it was a description field that had to have some input and could not be longer than fifty characters. To test those requirements with JUnit tests, we created two assertions to show that an error would be thrown.

A screenshot of a computer program

AI-generated content may be incorrect.

As we can see in the above code, the first assertion tests to ensure that an IllegalArgumentException is thrown if when creating a new task, the description is longer than fifty characters. Since the test is successful, we know that functionality works. The second assertion tests to see if using null as the input for a new task would trigger the expected IllegalArgumentException. Since it does, we know that the description field does what is necessary to meet the clients’ needs.

To ensure our code was efficient, we followed best practices, avoided repeating as much code as possible and avoided unnecessary function calls. Use of proper getter and setter functions help ensure there aren’t mistaken changes of data. In the constructors for contacts, tasks, and appointments we avoided repeating code by utilizing setter methods and checking it met requirements there, meaning if the requirements changed, we’d only have to change it in one place which can avoid an unexpected error.

A screen shot of a computer code

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.

Throughout this project we’ve used a few different software testing techniques. The primary type of testing used was component testing, sometimes referred to as unit testing. This type of test is “done in isolation from the rest of the system” (Hambling, 2019) to ensure the individual components function. This is done to catch errors early on before combining parts together where it might be harder to pinpoint the problem. This type of testing is great for functional requirements, as shown earlier with the JUnit tests for the different requirements for fields in the task class. Another type of testing utilized in this project was static testing, which is done without the execution of code. This type of testing “finds the causes of failures rather than the failure itself” (Hambling, 2019) making it perfect for ensuring the logic behind the planned implementation works before writing the actual code. This was extra beneficial during the creation of the appointment and appointment services sections where we had to utilize the depreciated Java Date format and had encountered unexpected results. By having already used static testing to know how it should work, which made pinpointing the problem an easy process.

There are many other kinds of testing techniques that we didn’t utilize but may be useful for the project down the road. An important next step would be integration testing where we “ensures that individual code units/ pieces can work as a whole cohesively” (*Different types of testing in software,* 2024). As more parts of the mobile application are completed, integration testing will be needed to ensure that these parts don’t interfere with each other. Afterwards we would want to utilize system testing to ensure the project functions as a whole. This type of testing isn’t just used for finding defects and ensuring functional behaviors but also is often used to “produce information that is used by stakeholders” (Hambling, 2019) in presentations of the team’s progress. Similarly, we can utilize acceptance testing to ensure we are meeting the needs of end users and stakeholders. This type of testing not only checks the satisfaction of our clients but also is useful for ensuring procedures are in place for other factors such as maintenance and disaster recovery.

While working on this project it was important to utilize caution. As a developer it is important to find defects early in the project where they are relatively easy to fix. The further along you are, the more time and resources it can take to make changes. As such, frequent testing throughout the process was important. It was also important to keep in mind the relationships with the code as you were testing it. As an example, when implementing the creation of a task, the checks to ensure the name and description met the requirements were done right in the setter methods, but for the unique ID you’d need access to the already existing tasks to ensure the ID wasn’t already taken. Thus, the necessity of this function is to search through the already existing list.

A screen shot of a computer code

AI-generated content may be incorrect.

This check gets used whenever a new task is added so it cannot create a task unless that ID is unique as per requirements.

Bias is an important consideration, particularly when testing your own code as it is much harder “to find your own mistakes – the so-called ‘confirmation bias’” (Hambling, 2019). It becomes too easy to see mistakes as criticism, so approaching testing with as objective a mindset as possible is important. During my testing of the appointment services, I hadn’t realized I had missed testing an entire requirement during the adding of new appointments until it was pointed out during feedback, something that would be even more challenging if you were the only one responsible for testing your own code.

Lastly, as a software engineer it is important to remain disciplined. Staying focused on finding problems early can save a ton of time and resources throughout the development process. Ensuring you are following best practices and avoiding shortcuts will ensure that not only is the code working as expected but makes it easier to read for other developers or testers that may be part of the project. The use of pseudocode and static testing are also incredibly beneficial to catch defects in a project before time is spent writing code. To best avoid technical debt, it is important to frequently review your code, test, and refactor as you go. Automated testing can be used to ensure “changes to the codebase don’t introduce new technical debt and provides a safety net for refactoring” (*What is technical debt: Common causes & how to reduce it)*. By being both cautious and disciplined, a software developer can prevent many defects throughout the software development lifecycle.

Works Referenced

*Different types of testing in software*. BrowserStack. (2024, August 22). <https://www.browserstack.com/guide/types-of-testing>

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing :*

*An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.

*Junit - Overview*. Tutorialspoint. (n.d.). <https://www.tutorialspoint.com/junit/junit_overview.htm>

*What is technical debt: Common causes & how to reduce it*. DigitalOcean. (n.d.).

<https://www.digitalocean.com/resources/articles/what-is-technical-debt>