Nicolas DeAssis

Project 2 – Summary

6/22/2025

In our project, we were tasked with creating a mobile application that enables users to manage contacts, tasks, and appointments. Alongside building the actual application's code base, we needed to ensure that the code was sound and tested. This was one of the requirements that the client requested.

Each functional aspect of our project had slightly different requirements that we needed to work on. Two example requirements our client requested were that the design of the contact service allows the user to create new contacts, so long as the contact ID didn’t already exist, and that when creating an appointment, we could not create an appointment for a date and time that had already passed. With each of the specific requirements our client gave us, we needed to make sure that we implemented the functionality and that the code was tested for correctness. We would carry out the tasks by first reviewing the client's documentation to ensure we understand what the client is expecting. When coding the client's software, we ensure that each of the parts ae built out correctly and works functionally. We then test the code to make sure that under different stress tests that it responds accordingly

When testing our code, we conducted an in-depth unit test of our code using Java JUnit testing. Since we created our code base entirely, it was generally easy to understand the logic that we created in the project. When testing the functionality and logic, we needed to create unit tests that would test different portions of the code. The idea is that we will need to cover all the branches of the program's decision tree. We also need to test invalid branches to ensure that our programs meet the client's requirements. We will know the amount of coverage code that our tests cover by running JUnit coverage test, which tells us how much of the code is being run through.

For all the functionality that we create, we make a test to ensure that we tests the code. For example, A screen shot of a computer program

AI-generated content may be incorrect.

This block of code is a part of the Contact class. It sets the contact's contactID, and when it does so, it makes sure that the contactID is valid according to the requirements given to us from our client. The contactID cannot be null, and the contactID length cannot be longer than 10 characters. With these two requirements, we need to test that the functionality works. We need to run tests that pass if, when a valid contactID is given, it accepts the input, and when a contactID is not valid, it does not accept the input and throws the proper error.

A close-up of a computer code

AI-generated content may be incorrect.

This bit of code is the respective test for the functionality we just saw. We test the contactID to make sure that when we are given an invalid ID we get the correct response. If the contactID is null or too long, the test will detect that the correct error has been thrown and that the contactID has not been set as the invalid input given. All the functionality that the project is required to have, as stated by the clients, has been tested similarly. The project is quite small, and we were able to do these unit tests for most of the functionality required. We did not create any other tests or functionality that was not required, so we know the project was made in an efficient manner. We then move on to our code-coverage documentation to show that our test covered a great deal of the project to make sure the testing was suitable as required for the client.A screenshot of a computer

AI-generated content may be incorrect.

This is the coverage test that we run to show which parts of our code the test was stress testing. We created different tests for different portions of the project, so when running all of the tests, we get our cover data to see how much of the code we can confirm works as intended.

Reflecting on the project, we can say that the testing techniques we used were white box techniques we are very knowledgeable about the code base so we are able to test specific parts using dummy data to ensure we are getting our results that we expect to get. When creating our tests, it is important that we don’t use bias as the creator of the code; we need to limit our bias by allowing the Junit software to test and give us the results of those tests. We need to create the tests in a manner that would allow us to force pass and get the value needed out of the tests.

Finally, we know we need to keep a disciplined mindset and test all aspects of our code and all branches of decision trees. We need to cover as much code as possible that is required for our project to meet client expectations. This means we need to be focused and careful to make sure that we are properly testing and covering all our code. Without discipline, we might leave in accidental bugs that reach our end users. Any bugs that reach out to end users will have unfortunately been allowed to escape and evade our tests and get to the end production.