In reflecting on the development of Project One (Appointment/Contact/Task and respective Service and Test classes). In regard to the software requirements, every test written was directly correlated to these. For each requirement, there was an assert testing it. For example, in contact service “The contact service shall be able to delete contacts per contact ID,” was tested via: Text

Description automatically generated

This test ensures that the proper contact is the one that is deleted.

Also, for the Appointment and AppointmentService classes I wrote tests to examine whether certain aspects were null, or appropriately named. Unique to the Appointment classes, I also wrote tests to make sure that appointments were not created for a date that was in the past. On top of this there are tests written to ensure that no ID is duplicated.

Graphical user interface, text, chat or text message

Description automatically generated

As well as testing the contents of the ID value, each and every method is tested with assertions making sure that the values meet the project requirements put forth in the documentation.

The overall quality of my Junit tests is sound and verifies all requirements put forth. When running as coverage we see all aspects of every implemented class being within the range of 80%-100% coverage (With the exception of one part of Contact being covered at 77%, this test is still being developed further to ensure higher coverage). Digging deeper in the coverage section we see almost all requirements being covered at nearly 100% levels across the board.

To ensure my code was technically sound, all requirement conditions have conditional logic and throw exceptions related to the specific requirements. For example:

Text

Description automatically generated

The updateAppointmentID method above throws exceptions for null values, as well as invalid length ID’s. To ensure efficiency I wrote tests for redundancy (pictured page 1, fig. 2). This ensures that we do not create duplicates.

Throughout this project we utilized unit tests. I have not performed any Integration Testing (Putting all the modules together, testing they’re working as expected one integrated. At least I don’t think I have!) or any Security Testing. These unit tests are absolutely crucial for everything from testing invalid variable values, to making sure ID generation is functioning correctly. In a professional setting, the sooner we test the sooner we will catch bugs and defects in the code. This can be the difference between life and death for a project and can sometimes lead to multi-million-dollar mistakes. Obviously with any project, testing is an integral part of development and absolutely will save time and money.

Reflecting on my mindset over the course of develop, I had to think like a tester. In the classes that I developed, like mentioned many are designed with the tests in mind. At first, I had been overly cautious making all variables and methods private. This caused me a lot of grief when I initially started writing tests, not knowing why nothing would work. Another important idea was to create constructors for every possible way that a user may want to input data (see lines 20-67 in Contact.java).

I found that the first set of tests that I wrote for a few classes were indeed influenced by some bias, with the pressure of deadlines I had not done my due diligence to ensure appropriate coverage. I had written tests that passed but had not dug into coverage enough to see that there were massive oversights relating to implementation of certain methods. As a result, I ended up re-writing tests in an effort to ensure a professional level of coverage. My old tests still exist as a multi-line comment where applicable. This allowed me to re-write without fear of losing the old work, while also having a clean slate to move forward with. I think ensuring appropriate coverage really helped me see the bias in my own testing.

Finally, discipline is incredibly important in our commitment to quality as a software engineering professional for so many reasons. If we are adamant in the pursuit of greatness with our code, we can ensure our client a secure, efficient, and technically sound product. We also reduce costs down the line and increase our value as a team member. When we cut corners (like in some of my initial unit tests) we introduce risk in all code produced afterwards. When we are diligent and test for all requirements, as well as possible conditions (for example lines 36-99 testNewContact() method in ContactServiceTest.java) we have covered all current and future issues that could relate to the newContact() method. This makes the newContact feature finished, preventing technical debt. Moving forward in my professional career, I intend to finish every feature I write, to ensure the prevention of future technical debt as a practitioner.