Dan Taylor

CS – 320

7 – 2 Project Two Submission

Unit testing is very important when it comes to building effective software solutions. It helps reduce the risk of running into program errors and improves the quality of the code by quickly finding defects and addressing them. The unit testing approach I applied for the three features were quite similar as I developed it to align with the software requirements. A major example was the Contactservice class which had some specific requirements such as, the Contact object shall require a unique contact ID string and it cannot be longer than 10 characters and it can't be null or updatable. The contact object was to require a firstName and lastName that cannot be longer than 10characters and shall not be null, they also wanted a phone field which is a string and must be exactly 10 digits and should not be null. Also, the contact class should also have an address field which should not be longer than 30 characters ad should not be null. The Contact object also has a service that does some crud operations with the contact object. All these were specific requirements for one of the three features, and all other features had similar requirements so the software must be implemented with those requirements to pass the unit test. The overall quality of my unit test was good as I tried to test each of the fields and the operation to check the validity of the test cases.

My experience writing the JUnit tests was agreeable. Prior to working on these projects, I had not ever written any unit tests, let alone JUnit tests, so this experience allowed me to learn something brand new and something that will be very useful down the line. My code is technically sound. The code I put together is modular, syntactically correct and utilizes the appropriate data structures for the task at hand. The following is an example from myAppointmentService class:

public void deleteAppointment(String deleteId) throws Exception{

for (int i = 0; i < appointmentList.size(); i++) {

if(appointmentList.get(i).getAppointmentID() == deleteId) {

appointmentList.remove(i);

break;

}

else {

throw new IllegalArgumentException("No such ID exists.");

}

}

}

This method utilizes the appropriate syntax for a for loop with an if-else statement within its bound and is modular with proper spacing and placement before the accessor method that returned the list of appointments.

One of the software techniques I used was the Dynamic testing method which involves the testing of the dynamic behaviors of the software code. I used this technique to check the behaviors of the various dynamic variables which are not constant and find the weak areas during software runtime. The other software technique which I did not use was static testing which is a way of testing the code without executing it. One major difference between static and dynamic testing is that identification of defects cannot be easily found by dynamic testing, such as development standard breaches and the detection of dependencies and inconsistencies in software models.

I had to utilize a learner's mindset while working on this project, as I had never written any type of unit testing prior to working on the code used in this project and I also had never written a project that needed to use multiple packages or objects. Caution was necessary, as I could not afford to let myself get carried away with the testing and ensure I only wrote tests for what was necessary. Once I finished the project, I had to take a moment to step back and appreciate just how complex it was and how I did it completely solo. I had multiple packages working in unison, even if some of the code for each object was similar, and that was something to be proud of because I had never done that previously.

I believe when it comes to reviewing your code, bias can be a major factor which has to be reduced, cause the person feels he is confident about the code he or she wrote and hence, won’t carry out proper testing of the units of the code. Like when creating the variables in a class, you might have the confidence since you were the one who created it, that the code is effective. We all should try to limit that bias when it comes to software testing.

It is very important to be disciplined as a software engineering professional, as not having the discipline needed could stand in the way of your progress and development as an engineer, preventing you from advancing in your career. Cutting corners while writing and testing code is detrimental not only for the company one is writing for or even their end users, but also for the developer and/or tester. Cutting corners to save time now could be very costly later, and you may end up having to start completely from scratch because you cut a corner and may have missed an important security feature. I plan to avoid technical debt by sticking to what I have learned and utilizing the best practices (comments, spacing, organization, secure data structures, etc.). Not only would I want to keep my job as a software engineer, but I also want to protect the organization that I am working for, and ensuring that everything is done correctly is a great way to do both.