**Project Two**

Elizabeth Danzberger

Southern New Hampshire University

CS-320: Software Test Automation & QA

Maylon Walker

20.06.2022

The approach I took when it comes to writing the unit tests for this project is somewhat straightforward. I looked at the requirements that were given to me and began thinking about how I wanted to write the tests, and what they needed to test for. For example, one of the requirements stated that the contact object should contain a string that holds someone’s first name, and that it should be required (meaning that it should never not exist), and that it should be no longer than 10 characters. Thus, one of the unit tests for testing the contact object should make sure that if a contact was created without a first name, an error is thrown; likewise, if a contact is created with a first name longer than 10 characters, an error is thrown. This is one way that I made sure the unit tests were aligned with the software requirements, and that the unit tests were relevant.

I believe the quality of the unit tests I wrote is good, despite having a coverage percentage of approximately 77%. Having a coverage percentage of 100% is sometimes infeasible, as this means that every single line of code in your program is covered by a unit test. For this project, and for many others, I don’t believe this is completely necessary. An ideal coverage percentage should be tailored to the program and its intended functions (but of course, you would ideally aim for a high coverage percentage). The unit tests I wrote for this project cover the core functionalities of the program, including making sure that each class is instantiated with valid parameters and that each method for each class throws errors when needed. All in all, this is indeed the goal of unit testing. In addition, I spent a lot of time changing values and messing around with the unit tests, and they succeeded in catching everything I threw at it. Thus, I feel that the unit tests, although they do not cover 100% of the code, are effective.

I believe I’ve stated this in a previous assignment, but I will consider code efficiency as a subset of “technically sound” code, so I will answer both of these prompts together.

In each class (Contact, Task, and Appointment) there is a find() method. This method provides an easy solution to searching through the currently existing contacts, tasks, and appointments in order to find the object with an ID that matches that of the ID passed to the find() method. It returns the index of the matching object in the corresponding ArrayList, or it returns -1 if the object was not found. This method is used throughout the project to determine if an object with a given ID already exists (each object should have a unique ID), and to get the index of the object if it does. Although the search algorithm is a simple linear search algorithm and would be quite slow for an application that contains hundreds of objects stored in memory, I believe it is an efficient way of quickly finding an object in an application like this (if the objects are stored in memory, ideally there would not be many of them, as using up a lot of memory is not viable and is typically considered bad practice). I believe the code is efficient in this way, and is therefore at least partially technically sound.

Throughout this project, I have mostly employed static and dynamic software testing techniques. During the static testing phase, I manually reviewed the code in order to make sure that the code made sense and was in alignment with the software requirements. I ensured that the data structures were implemented properly, and that the code was error-free and that exception handling was properly implemented. After making any edits, I again made sure that it aligned with the software requirements.

Some other software testing techniques I employed were unit testing (what a surprise). Unit testing was the main purpose of the milestone assignments. Unit testing involves writing tests for small chunks of code, like individual methods. It allows you to focus on testing one particular aspect of the code, and to limit your scope of focus. This can help to make sure you think about one aspect of the code really well, and cover as many bases as you can. It also gets developers into the testing mindset, and helps them incorporate testing into their daily programming work. It also sets you up for some other software testing techniques…

Integration testing, system testing, and acceptance testing are other software testing techniques that are extremely important. Although I did not perform a lot of these due to the small size of the project, for a larger project they are a necessity. Unit testing allows you to test components, and integration testing ensures that they all “play nice” with each other. System testing involves testing the system as a whole. Acceptance testing is where the “dirty” and “true” testing happens, as this is the part where you release your program to a subset of users in order to get an idea of how it will behave in the wild. This gives users the opportunity to provide feedback, and to discover any faults that were not noticed during the initial development and testing of the application. In an ideal situation, you’d perform all of the software testing techniques and more.

Thinking about how I employed caution throughout the project is difficult. I am again not quite sure what is meant by “caution,” but I will interpret it in my own way and answer appropriately, which I am sure is what you are looking for anyway.

Throughout this project, I had to think carefully about how I wanted to implement the unit tests and how I wanted to design the data structures used in the application. Of course, the software requirements provided a great start, as the rubric outlined what the application needed to accomplish; however, how I wanted to implement these things was the real question. I had to exercise caution here, because I could not just start writing code and fit it to the requirements later. That’s certainly not the way to go. I had to think about how I wanted to design things before even writing the code, as is typically considered good practice.

In my opinion, the code written for this project was not very complex, and we were not actually required to ensure that each of the classes integrate well together and form a functional application, but I am nonetheless certainly able to appreciate the complexity of such an application. I’m not really sure what specific code to point to in this regard.

Bias is a really interesting thing when it comes to being a software developer. When you are writing code, you may of course be reasonably humble enough to expect that you will not write error-free code. Even so, it is important to have others review your code and provide feedback from their perspective; everyone has their own programming style, and you may not be aware that some of your code can be improved upon, or that some things could have been implemented differently (perhaps more efficiently). Testing your own code is where bias can really expose itself, as you may test your code in a manner that aligns with your coding style or thought process, which can mask bugs. When testing, you really need to think of ways that your code can break or function inappropriately. There are many cracks and crevices where bias can hide in the realm of software development. Allowing others to provide a fresh perspective on your code can greatly reduce the negative effects of bias on your program.

Being a disciplined software engineering professional can sometimes be really difficult. It is important to take accountability for the code you write, and to always put forth great effort. Cutting corners is not an option. It can cost you or your company your reputation, career, money, and perhaps the greatest loss of all, time. Personally, I plan to get as much feedback as possible from my peers regarding my code in order to learn as much as possible. In a professional environment, I feel that a lot of learning can be done from your peers, and whether you learn something good or something bad. Hopefully you can spot the difference.