**Project 2**

**CS-320-12555-M01 Software Test, Automation QA 2024**

**Kenneth Pinkerton**

**Summary**

* Describe your unit testing approach for each of the three features.
  + To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
    - As I was going about creating unit testing for each of the features I looked at the requirements that each object would need to pass and used these exclusively to create my tests, I tested first that the program could properly determine what a good object looked like and next created a series of test to test each point of what would be an invalid input, narrowing down from what passed and only changing the section that would be a bad input so I could more easily determine what specific area was failing or needed correction. An example of this is from my TaskTest.java file for a good conustructor
    - A computer screen shot of a program code

      Description automatically generated
    - And then a test of a task ID that is too long.
    - A computer screen shot of a program code

      Description automatically generated
  + Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?
    - I know that my JUnit tests were of good quality because all of the tests passed and that both good constructors were tested as well as each requirement for all objects. Items like the ID for an object cannot by over 10 characters, and the ID cannot be null were broken down into separate tests. For items that require more logic these had specific tests such as testing for phone numbers required a number to be entered that was exactly 10 characters long. The minimum tests are covered however more in depth testing would use regex or an api to lookup the number that was entered to make sure that it is a valid number and not something like all 9’s. I feel the quality is good to meet the requirements here, but actual production code I would want to include these in to prevent errors, duplicate phone number or address were also not considered, but were not against any requirements so I did not test for these cases.
* Describe your experience writing the JUnit tests.
  + How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
    - As I was writing the tests I would actively write the test, run it to insure that it ran correctly and if it failed where it should have succeeded or vice-versa I would narrow in on the area that was being tested to find what the source of the error was. Because many of the objects were very similar it was easy to think that I could just pull in from other objects in the same project and change a few of the requirements or names and it should work as expected, however there were some errors in this as there were some areas that even though they ran right because the IDE had automatically changed the name or suggested what it thought would be the correct name it was off just enough to cause some erratic behavior. For instance when I was working on testing that the data for the Appointment.java was not for a previous data I spent some time trying to return a variable that equated to true or false, but the variable itself started out as null, this triggered another test as this could not be null. By working through the tests though I was able to correct the behavior and get it back on track. (Appointment.java lines 24-50)
  + How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
    - One of the ways that I tried to keep the code efficient was to limit the complexity of the code by having the functions only performing a single task where appropriate as much as possible. For example, in Appointment.java line 15-21 I have the program testing that the object entered is both not over 10 characters long and is not null as these both are able to be combined into one simple check as a return for a helper function. This helper function is then referenced whenever a new object is created or modified, if there is a problem with the entry the program will not allow for invalid inputs. In this I also helped by keeping the code DRY so I’m not calling each input that is valid/invalid to check against instead it checks what is at the boundaries as much as possible to reduce the runtime and keep the code efficient.

**Reflection**

* Testing Techniques
  + What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
    - One of the main techniques that I employed in this project was white-box testing, and automated testing. Since I knew exactly what the structure of the objects was the Junit testing used this knowledge to directly test exactly what would be good or bad inputs. As I did not confirm by hand, but instead used an automated software system that ran the tests for me this helped to speed up the testing process and ensure that each of the tests was done the same each time to prevent human error from slipping in more into the tests than if I had verified all of the inputs by hand, if that were a possibility with this project, since there was not user interface to input information this was not even an option to test each by manual entering the information.
  + What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
    - Some other testing techniques would be black-box testing and manual testing. Since this project did not have any user interface to use there was no way to test the interface from the users perspective like using XML or tools like Selenium to test inputs that a user would click on as they are traversing a website. Manual testing is another example, but would involve a tester reviewing the layout of a site or application to make sure that it meets certain UI requirements among others and that it matches what the client is looking for in the delivered software solution.
  + For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
    - White box testing allows you to test the structure of objects and various inputs that you know exactly how they should look like and how the system should respond this works well for most projects that you are developing and can help to prevent errors from being entered into the system and can help increase the security of the program as well as it can prevent certain types of injection attacks from being used.
    - Black box testing allows you to test the system without having to know exactly what it’s doing under the hood. This can be good when you’re designing UI’s for users and need to make sure that it’s accessible to a wide audience and doesn’t have issues like viewing trouble for those with color blindness.
    - Automated testing allows you to perform a set test exactly the same way each time it is run, this can help when you know exactly what you need to test for and may need to test a large number of different types of input. This would be good to use in most cases and can help in preventing possible inclusion of human error into the testing as long as the tests themselves are well structured. While it’s still possible to have human error with automated testing it lowers the risk significantly.
    - Manual testing helps when you have systems that would be difficult to replicate using automatic testing and can help with reviewing the code, or logging into systems, or replicating a known error, but you don’t know the exact cause of the error. This is also good to use on most types of projects as this can be more thorough without knowing the exact test that you are testing for, or in other words debugging.
* Mindset
  + Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.
    - In approaching this project I felt that I was somewhat cautious, however not quite as much as if I knew I would be testing code that would be launched into space, in medicine, or with finances. I strove to start off with the most basic version of the code that I could and slowly add tests and features as I progressed. I felt that it was a more solid approach than trying to code everything first then write all of the tests and finally test everything and hope for the best and spend more time hunting down bugs later than iteratively build the solution from the ground up piece by piece and look that each piece was in line before proceeding.
  + Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.
    - I think that as I’m still newer to development I still don’t fully trust myself to write code without checking that it works, I think my approach of starting small and slowly building onto it lends itself to that line of thinking. I can see where others if they know that something has worked in the past they may just take what worked and paste it in, making only the changes they need to. Admittedly I did this to an extent, but when I did catch myself starting to do this I pulled myself back and instead went back to checking the code line by line as I entered it in to make sure it was solid before proceeding to the next section. I think this kind of highlights bias that you feel your code is good just because you wrote it and it worked before, but you have to be careful that what worked before, may not always work.
  + Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.
    - To me being disciplined in programming can be very important in some cases lives may be on the line if your code doesn’t live up to it, you end up being the one at fault, just the thought that my mistake could lead to someone else’s, or my own, tragedy like that is more than enough reason to give pause to me. While not all projects are life and death, the projects and programs that we help create could be used to help or hurt others in their lives, if for instance we are working on harmless app that becomes a popular game, but has access to customer information if we don’t test the program enough and allow hackers to compromise the system because we just happened to be lazy when we were implementing the systems, we might have known there could be problems, but we didn’t act on it we could be liable, and those people would then have their information stolen and have to try to fight that intrusion because we wanted to try and get the project out a little bit quicker.