**Project Two**

Jeremiah A. Boothe

Department of Technology, Southern New Hampshire University

CS-320-H7028: Software Test Automation & Quality Assurance

Omar T. Lopez

December 10, 2023

# 1. Summary

## a. Describe your unit testing approach for each of the three features.

### i. To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

My test approach was fully aligned with the requirements. The biggest hurdle was getting more familiar with the capabilities, limitations, and syntax of Junit. In a few more projects I will have the test count slimmed down to just what is necessary to hit the coverage. My service class parameterized test(below), took care of the null and length checks. I provided one failed argument, and the remaining arguments were set up to pass normal conditions. The single failed argument was a check that it would not throw errors as it went through the arguments and saw ok that passes, that passes, oh that one fails, never mind no entry allowed. I also threw in a duplicate entry check for the id 7.

A screenshot of a computer program

Description automatically generated

Here is an example from my TaskTest where I apply the positional shift to updating, of course there is nothing to call to update the ID, so there are only two positions to update.

A computer screen shot of a program

Description automatically generated

Then to further have a visual confirmation, I called my print map functions to double check that the entries that should have passed did in fact make it into the map, and the ones that should have failed did not.

A computer screen shot of text

Description automatically generated

**The following would sum up my internal thought process, in a single test. With my error handling process everything was reduced to:**

**1. does not throw error and is created and added**

**or**

**2. throws an error and is not created or added.**

**A computer screen shot of a program code

Description automatically generated**

### ii. 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 have mixed feelings about this, it falls in line with a quote from the Software Tester Foundations book, “**Testing shows the presence, not absence, of defects.” (Hambling, 2019) This was a lesson I experienced firsthand as **there were several times where I was passing tests, but the debugger showed me that the code was not doing what I thought it was despite the 100% coverage and tests passing. 100% coverage gives me about 50% confidence in my code.**

**A screen shot of a computer

Description automatically generated**

A screen shot of a black background

Description automatically generated

A screenshot of a computer program

Description automatically generated

## b. Describe your experience writing the JUnit tests.

### i. How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

#### Debug

The best thing about debugging is that it is about as close to watching your code execute in real time as you can get with compiled languages. One of the things I miss from Python as a scripting language is the real time debugging, where you could watch the changes in real time with no break point pauses. One of the more fun parts was once it is loaded into memory, you can write command line functions and watch the values change as soon as you hit enter. With compiled languages this is close if you choose your break points well enough. Sometimes I will put unnecessary continue statements, because they are a great breakpoint to capture what just happened within a function if other break points are not doing the job.

A screenshot of a computer program

Description automatically generated

#### Security

My best security effort was through keeping methods private or at most package private. I only open things up as public on an as needed basis. I also focused on strong encapsulation and separation of concerns between objects and their functionality and even between methods within a class. The boundaries set for null, and length are also a level of security whitelisting acceptable values and rejecting anything outside those boundaries. From the UML I consistently achieved aggregation and avoided inheritance. Inheritance on its own is a great way to extend functionality but in larger projects with a lot of inheritance it can become difficult to keep track when you’re 15 inheritances deep and making changes to functionality, major issues can arise, there is more flexibility in aggregation with less rippling effects, but it can also be a bit more complex.

A screenshot of a computer program

Description automatically generated

#### Error Handling

I had extensive error handling functionality with appropriate throws for each error thrown by the appropriate layer and allowed those errors to pass through to the test level to be handled and acknowledged. This helped me pinpoint and resolve issues quickly and ensure that no errors were handled before they were acknowledged appropriately, no errors lost in the oblivion.

For Example:

1. The constructor for Task is set up to throw the error up the chain.

A computer screen shot of a program code

Description automatically generated

1. When a function called by Task, such as the length check or null check, or even the setters, throws an exception, the constructor terminates and throws the error received from the failed functions up the chain through the service class.

A computer screen shot of text

Description automatically generatedA screen shot of a computer

Description automatically generated

1. With the thought process of throw or pass in mind, I designed flow of error handling to throw when an error occurred at a particular level, and to let that error pass through. The Services throw service specific errors, and the object classes throw object specific errors.  
   A computer screen shot of a program code

   Description automatically generated
2. That allowed me to capture all the errors at the test level and include them in my testing process effectively, without losing errors because they were handled at a lower level.  
   A computer screen shot of a program code

   Description automatically generated

#### Incorporated Instructor Feedback

1. This was a big one, you leave some great feedback, and my code is all the better for it. I found my own style and implementations, but each point of feedback helped me find the target.

A computer screen shot of text

Description automatically generated

1. The output of my tests is so much cleaner now, after applying your feedback about names and general improvements it put me on the path to make them something enjoyable to look at. In the past year I have made a lot of improvements on my code structure, naming conventions, and commenting, still some room for growth but now I am applying that to unit tests. It is not nice to give someone who is reading your code a headache that makes them feel nauseous, rude even.  
     
   A screen shot of a computer screen

   Description automatically generated

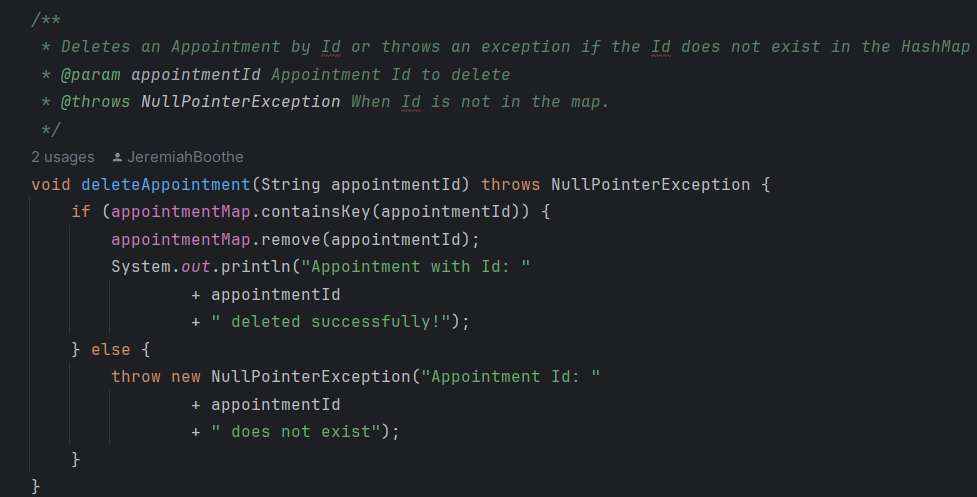
### ii. How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.

#### Efficient Code

1. Hash maps are an efficient data structure with rapid access to an entry so in execution speed it is an excellent choice.



I used comments to tell exactly what functions do and kept logical naming conventions for readability and ease of understanding.



1. I am always trying to apply the SOLID principles as I write code. The style I am trying to develop includes bite sized functions written in a way that can be reused to reduce code redundancy. The less any single function does the better. Even still, I am not entirely a purist and sometimes it is ok to maybe have a function do two or three things if those tasks can be grouped logically. Then again, that sentiment changes as the size of the code base grows, the larger the project the more purist I become because the likeliness of reusing a specific one purpose function goes up.

A computer screen shot of a program

Description automatically generated

1. In CS-210, my instructor emphasized the exit early principle, his example was more about the effect of running branching paths on performance. Since then, I have worked to adhere to that. Fail and exit as early as possible when executing any process.

A computer screen shot of text

Description automatically generated

1. Avoided Hard Coded Values throughout.

The setter above is a good example. A few places where I could have improved that, is in the tests themselves, I did implement some variables to use in place of hard coded values within the tests and defined them at the top level of the tests, but I could have done a little bit more of that. For the future, I am going to experiment with pulling the values for my assertEquals directly from the properties of the production class itself, so I do not have to keep editing the tests to reflect changes made in the production classes. If Contact has a String errorMessage, and ContactTest assertEquals expects the exact contents of errorMessage, then why not just reference errorMessage in Contact from ContactTest to make sure that is what made it to the test?

# 2. Reflection

## a. Testing Techniques

### i. What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.

#### Static Testing:

##### Dry run

I spent a good bit of time doing this, usually at the beginning of each session when I would first open the project, I would read it while it was fresh, and I had not lost objectivity to spot any issues that might pop up. Sometimes I found errors and fixed them as a result, including changes that I had missed previously. One error was that I had not fully formed a function call that resulted in the created object not being placed in the map after creation.

##### Reading requirements documents and planning

All my planning and execution was built around the requirements of the assignments. I did exercise a little leeway with my interpretation especially in the contact milestone where I wanted to try out my idea of a second constructor and had fun implementing the automatic ID generation. In every course, there is a feeling out process in the first couple of weeks to understand how each instructor grades and what they expect and adjust. Some instructors are a bit stricter than the rubric; some are word for word by the book, and others give a little more leeway to go off the rails with off script ideas. I never strayed too far and once I got some feedback; I adjusted my scope.

##### Feedback

I know I mentioned this before, at least in a message along with my submission, but you have given the best feedback I have received so far. It helped me dial in on the requirements and your expectations in a positive way that was energizing. It is a great model to follow for any future situation where I end up being the one giving feedback to someone else.

##### UML (Unified Modeling Language) Diagrams

As you know, I’m a fan of UML, it gives a lot of information in a very concise way, you know what types you’re dealing with and the names of the methods and properties, and that’s really one of the most challenging things about programming, knowing what types are already handled, and figuring out how to transition from one type to another so that data can be used with something else. It is somewhat of a black box in that you do not really know what is going on inside, but you know it takes something and returns something, sometimes it is the same type that went in and sometimes a different type comes out. It shows you the ruleset you are working with, for example, if it is an abstract class or interface, you have a subset of options for how to implement those vs. a class that can be inherited. While I do not shy away from referring to developer documentation, I prefer to minimize scrolling up and down a 50-page document as much as possible. I would rather read the name of a function, understand the type and the rules that I am limited to and then go to the developer documentation if I need to know more.

#### Dynamic Testing:

##### Unit tests

This was the heart and soul of this class for me. The Software Testers foundation book was awesome, I learned some great concepts to apply to unit testing, but unit tests were something I had put on the back burner for a few years and am glad I finally got the chance to make them my primary focus.

##### Component Integration testing

I arrived at this by accident. It never crossed my mind until I ran with coverage the first time. It was one of those aha moments where I was like oh of course testing the functionality of the service class that is directly working with the object class would also test that object class. This is demonstrated in my UML above which shows the flow and interaction of the classes.

##### Regression Testing

I was hitting the rerun button on tests as quickly as I was changing the code to ensure that I was in fact improving it instead of breaking it. I have tried to find a run count for the tests within IntelliJ but so far, I cannot find a way to view that. It would be interesting to have a recorded quantity of test runs to include, but I rarely went more than a few minutes without rerunning the tests after I was confident, they were operating correctly.

##### Error guessing

I was guessing at errors to get through much of the contact milestone, once I got my error handling flow down, I no longer had to rely on guesses because I had errors that pointed me to exactly where the problems were.

##### Exploratory:

I applied exploratory most fully during the Thanksgiving holiday by time-boxing and prioritizing exactly what I needed to accomplish in the time I had available.

##### Checklist-based testing:

Checklists are something that has been part of my process since before I even knew it was considered a type of testing. To help me stay focused on the assignments I copy the requirements and rubric into my projects and comment them out then when I catch myself getting too immersed in something I pull back and go over the list. Then I delete them as I accomplish them until the entire comment is gone. I started doing that because I got tired of spending the time logging into Brightspace to check.

##### Statement Testing:

While I worked through my tests, I initially covered all methods but not all conditions within each method. Through the testing process and referring to run with coverage I could see that 100% of the methods were being executed but a lower percentage of the lines were being accessed. To remedy this, I had to adjust my tests to trigger the exceptions within those methods. The activation of a method does not ensure that all the code within is executed, especially when applying the failure and exit early principle and general organization of internal code to avoid running more code than necessary in any given execution.

##### Decision Testing:

My thought process behind testing was very Boolean logic and truth table oriented. The Boolean logic handled the null or not null, and too long or not too long cases, and the truth table approach helped me logically check argument positions against null and length.

##### Black Box Testing:

My testing process was focused on requirements, and since black-box testing covers everything, requirements based, all my testing would fall into this category. Once the tests were written to cover the requirements, I could virtually ignore them and work on the production code and just run the tests to confirm I was meeting the requirements.

##### White-box testing:

I used this with all my structural testing such as unit tests, and component integration tests, even the flow of error handling and values through the service and non-service classes would fall into this category of testing.

##### Boundary Value analysis:

The boundaries defined in the requirements came in the form of null and length. My tests and production code were written to account for that. Objects were outside boundaries when they were null or longer than the specified length, and within boundaries when they were not null and not longer than the specified length.

##### Decision table testing:

I applied this heavily in my parameterized testing, the positional failing of each argument while having passing cases in each other position to ensure that any single failing would result in the whole object being rejected. The objects required all positions to be true or have valid entries for the resulting object creation to be true.

##### Use Case testing:

I put myself in the seat of a user and approached the behavior with my own expectations and experiences when using applications. The inclusion of time in appointment class is a prime example, as it is hard to see a purpose behind an appointment date without a time, and I imagine any user would immediately ask, “why can’t I input the time of the appointment?”

##### Equivalence partitioning – Input Partitioning:

I grouped checks in null or not null, and too long or within range. A value exceeding the length was rejected and a value within the acceptable range was accepted. This was applied to each positional argument available during creation and updating of each object/map entry. As demonstrated in my parameterized tests above.

### ii. What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.

#### Equivalence Partitioning – Output Partitioning & Other:

Equivalence partitioning was handled on input with the grouping of null or not null, and too long or not too long. It was not really applied for output directly but more because of the limitations on input and the inability to exceed the input when displaying data.

#### Integration testing:

##### System Integration:

There was no system integration testing since there was no database or interface, and each milestone was an isolated piece that was not integrated together into a more complete system.

##### User Acceptance testing

No user acceptance testing, but instructor feedback filled in that gap.

##### Non-Functional Testing

Security was not tested but elements of security were considered to the extent possible, I made things private and package private and tried to maintain encapsulation throughout the project.

### iii. For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

As a tradesman, I have long had the “Measure twice cut once” mentality to things and doing so has never hurt and many times during that second check I realize you got something wrong and fix it before you make a mistake that requires more investment and time. I have never cut fabric or a board too short and saved time or money. Software engineering is complex but the foundation of the entire process of testing is rooted in that mindset. Each type of testing is a unique way to measure, but unlike a good tape measure, testing does not show that something is correct, only that within that specific measurement of functionality and quality, no errors were found. The process of software testing and the necessity of applying so many methods are a bit overwhelming at times but considering the examples we have gone through over the term, the criticality of as close to perfection as possible can be examined in loss of life, finances, and reputation. The most uncomfortable part is that in many of those examples no number of processes can eliminate human error, laziness, negligence, or greed. It is important to get it right because technology and software control the world and with slight mistakes people can lose their life savings, planes can fall out of the sky and nukes could be launched. In the best-case scenario, bad practices could result in extra expenses and annoyed customers taking their business elsewhere and telling all their friends to avoid your services.

## b. Mindset

### i. 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.

**There was no caution applied at any point for the projects. I am a student learning and trying to improve my skills and to do that in a meaningful way I need to run with ideas and inspiration as they hit me. What you saw was the neater cleaned up version but there was quite a bit that I deleted, some of my ideas did not work out, a bit more time working on them over a week or two and I could have made them work out. One example which is now deleted was a regex function I was working on for the appointment service. It was the first idea I had for how to handle user input, but time was ticking, and I had to use a more direct method that I could pull together in a reasonable amount of time. The idea was too big for the project and time limit, it would have allowed a few common date formats as input and parsed it into the correct format for the date time check. So, while caution was not part of my process at all, the weekly time crunch was. I enjoy going too far to the point that the whole project is completely broken then trying to see how fast I can find and fix the problems.**

The complexity of the relationships between the classes is what inspired me toward the pass/fail throw or do not throw mentality. For about the first half of the contact milestone it was easy enough to jump around and toss out some ideas, but once the tests started coming together and I was 4 files deep I was slowing down a lot more and starting to feel some of the frustrations of endless context switching. So instead of continuing that path I got my error handling in line to help me pinpoint where the problems were, not necessarily the exact spot but it was easy to trace a thread, with decent separation of concerns I would just follow the thread and fix the issue.

### ii. 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.

**In the contact milestone I was getting so wrapped up in trying out the features of Junit to gain a better understanding of how it worked. After many hours of constant back and forth between the production code and the tests and I lost objectivity and ended up writing tests to pass code that was not functioning as I originally intended. It is not a new thing for me to lose objectivity, I have experienced it many times in my years as an upholsterer and a musician. As an upholsterer after 16 hours on a project with the bulk of that trying to get the finer details just right such as strip matching and pleat sizes, it’s hard to immediately pull back and see the bigger picture and I often couldn’t really appreciate that bigger picture until I saw the piece again after a few weeks of not being fixated.** **As a musician, there is an often-spoken rule, never mix or master your own music, at least not without a few weeks break from the songs before making the attempt. You will get so wrapped up in how you hear it in your head and through tracking and editing, that your ears are no longer reliable because that is getting warped by your brain. It’s so dramatic that you can mix a song and go to bed feeling awesome at how great your mix is, and a week later listen to it and have your heart drop to the floor at how terrible it sounds because you weren’t mixing what you were hearing from the speakers you were mixing what you were hearing in your head. I can very much imagine what it is like to lose objectivity. The three milestones and moving between testing and production code was the first time I truly experienced the loss of objectivity as a programmer. This was particularly apparent in the contact milestone but by task and appointment there was a lot less context switching between unit testing and writing production code.**

**A more effective approach for the future would be to break down the logic behind the requirements, and write the tests from that, then sleep on it, and come back the next day or a couple days later and write production code with the same logic and requirements in mind while not thinking about the tests themselves or even looking at them to whatever extent that’s possible.**

### iii. 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.

**The reasons that are most important to me, beyond my own drive to always do my best, come down to lives, livelihood, and ethics. People die over low-quality software, they lose their entire life savings, their jobs, and cutting corners is dishonest and unethical. While I am always striving to do my best, this term really informed me of just how critical of a job software engineering is. Having QA and Security together really shook my world view of being a programmer, not in a way that I find discouraging, but more like I must really step it up to some exacting standards because an oversight could mean hospitals stop functioning or people get the incorrect medication or dosage. It was not that I was completely unaware of that, just with my limited experience of playing with apps and Unity, I just never imagined myself being in a position where my work had that much gravity, and at worst people would be annoyed that they could not log into their game for a few hours. In part because I was aware my skill was low, and no one in their right mind would have hired me for anything critical. Things that felt impossible or incredibly difficult a year ago are simple routine tasks now, and looking forward with everything I have learned, it is plausible that I will be working on something critical. My scope is finally shifting from that of someone that writes some code to that of a software engineer. Those are two vastly different mindsets, which I am just starting to grasp as I type this.**

**My plan to avoid technical debt as a practitioner in the field is to do what I have done through this course, internalize feedback, and listen to people that are far more experienced than I am. There is no room for me to have a chip on my shoulder and think I know better than someone else, but that also becomes more complicated in that there may come a time where I must speak up and fight for something when best practices are not being followed whether intentionally or unintentionally, and not falter in my conviction with my livelihood on the line. Humble and stoic, confidence without hubris.**  
 **For now, I will keep learning and improving my skills, this includes separating the writing of tests and production code and spending more time planning before I even write the first line of code. I try to keep SOLID principles at the forefront of my mind as much as I can manage, and over time I have been analyzing idiomatic programming, pure functional programming, as well as design and architecture patterns. My thought process is aimed toward ingraining proven frameworks for problem solving into myself. And while not much caution was applied to the projects, I was experimenting wildly with as many of these concepts in my mind as I have been able to process in a meaningful way so when I do end up in a job environment, I will have the best practices at my fingertips and experience to back it up. Here are five books from my personal collection, that have been and continue to be guides along the way.**

A group of books on the floor

Description automatically generated

# References

Hambling, B., Morgan, P., Samaroo, A., Thompson, & G., Williams, P. (2019). *SOFTWARE TESTING: an istqb-bcs certified tester foundation guide.* (B. Hambling, Ed.; Fourth). Bcs, The Chartered Institute.