**Module Two Journal**

Bradley Brennand

Southern New Hampshire University

CS-320 Software Test, Automation

Professor Haney

March 28, 2025

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

The assignments listed specific requirements for the contact and the task class. Specifically asking for the contact and the task to not be updateable, be a maximum number of characters and to not be empty. The classes that were created covered all of the specific requirements to ensure they were met. The tests that were created to test these requirements also ensured that the parameters of the requirements were fully met. To verify the requirements were met, I wrote test code to specifically test and make sure something was entered into the field. I also verified that the information entered in the field was not greater than the required number of characters allowed.

*if (firstName == null || firstName.length() > 10) {*

*throw new IllegalArgumentException("Invalid first name"); }*

Looking at this code, it says: if the first name is empty or the length of the first name is greater than ten characters, the program will display an error message letting the user know that it's an invalid first name.

**Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

After the JUnit test was created, the tests were run to ensure that requirements were met. Running the test and looking at the level of coverage ensured that the entire contact and task were tested to one hundred percent. This level of coverage ensures that there are no errors and the level of code written is performing without defect.

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

Technically sound refers to specific criteria and well-constructed. Following the criteria that were given and writing code to meet those needs set by the client is how I made sure the code was constructed to meet those needs set forth. One of the criteria was to be able to add a task in the program. I not only wrote the code to add a task, but it was written in a way that adds the task and verifies there is not redundancy.

*public void addtask(task task) {*

*if (taskids.containsKey(task.gettaskID())) {*

*throw new IllegalArgumentException("Task ID found");}*

Looking at this code we can see that an argument was added in to ensure that if the program did not find what was being asked that it would throw an error and let the user know that it was not found. This would allow the user to try again or use a different command instead of the program getting stuck or breaking. Adding a set of checks and balances is a great way to ensure a program's code is technically sound.

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

*public void addtask(task task) {*

*if (taskids.containsKey(task.gettaskID())) {*

*throw new IllegalArgumentException("Task ID found");}*

Looking at the same code we can notice the Illegal Argument Exception; this represents an error check in the program. In this aspect adding a task and checking to see if the task already exists by looking for the task ID. This is specifically checking to ensure that the task that is being added doesn’t already exist in the program which eliminates task redundancy. Reducing code redundancy and writing code efficiently is a great way to ensure efficiency in a programmer's code. This example of code is short and specific. Keeping code simple can be another great way to keep code efficient. Complex code can be harder to modify and integrate into other programs.

**What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

There are many different testing techniques that could be used during testing of code. For each of the modules that I coded, I used JUnit tests. Also combining a set of static and dynamic tests to ensure all code gets covered. The JUnit tests that were used were to identify any errors written in specific functions of the code. These JUnit's test each aspect of the functions that were written. Static testing is a technique used to identify errors in code without running the code. Using things like the compiler that annotates errors during the writing process of code. Dynamic testing is another technique used to check for errors but this time it runs the code and identifies errors after the code has been written. The type of dynamic test taken would have been the coverage. Running the code compiler and looking at the level of coverage that the test case was identifying in hopes of reaching one hundred percent coverage. This states that all functions of a code have been tested for complete coverage and there is no room for error.

**What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.**

Other testing techniques that were not used could be manual testing. Manual testing consists of running code and manually entering inputs to see if the program handles and operates the way it was designed. This process can be extremely time consuming as we would have to input different values multiple times to retrieve all expected outcomes. Integration testing is another technique that was not used. Integration testing is a technique to test how different functions of code all work together. This is how much larger programs are built.

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

Let's look at a calculator for instance. There will be add, subtract, multiply, and divide functions for a calculator. Putting them all together under one program can be tricky. This is where integration testing comes in handy. In modules three and four the JUnit tests were used to test the boundaries of the code. For example, the requirements stated the input should not be null, and it can’t be more than a specific number of characters. I wrote a test case in the JUnit to specifically test that the input was not empty and not beyond a certain number of characters. These are good examples of how to use the JUnit tests. Static and dynamic tests are similar. Statically looking code over for errors. Dynamic testing can be more efficient. Dynamic tests also allow us to see the level of coverage our test cases are covering to ensure all requirements are being fully met for the program's completeness. There are many different programs in today's world and each program could be tested in a different manner. It is left up to the developers and the testers to ensure they are using the right techniques for the programs that they are testing.

**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.**

Acting as a software tester, I only wanted to test the specific criteria that were absolutely necessary. There is no need to write more code than is necessary in such I aimed to keep it simple. The contact and the task functions were very similar; therefore, I was able to create the first set of tests for the contact function and then copy that code and minimally modify it to apply it to the task function. This was more efficient and less time consuming during the coding phase of the project.

**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.**

During the writing process of the code, I tried to limit my bias by remembering that I'm not perfect and I'm bound to make mistakes. Remembering to learn from those mistakes can make anyone better. As a developer I not only want my code to work but also want to ensure it's safe and free of defects. If I was a twenty-year software developer my ego may create bias, but always trying to stay humble and knowing that technology is always evolving could keep that level of bias minimalized.

**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.**

Continually learning technology and techniques is a great way to stay disciplined. Proper testing techniques and thoroughness while testing can be a great way to ensure your code stays safe. This will minimize program failures and flaws that could cost companies money. Some companies like hospitals depend on certain databases and programs that maintain medical records and medical history. Should these types of programs crash it could be devasting and put people's lives at risk. Without someone's medical history hospital staff wouldn’t know what medications a person was taking or what medications they could be allergic to. This is one extreme case where a software developer would need to be extremely disciplined to ensure the safety of their programs.