Evan Carter

CS-320-R4840 24EW4

**Summary and Reflections Report**

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.    **The tests implemented for each class were directly generated from the requirements. Given a requirement such as “the contact service shall be able to add contacts with a unique ID,” tests were created to cover the full range of possible outcomes: verifying that the add function works as intended and that given a non-unique or null ID, a handleable error is thrown. This process was repeated for each requirement individually, to achieve maximum coverage for all cases across each class.**
      2. 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?  **The high coverage percentage that we have (80%+) shows that our tests are incorporating a significant majority of the code with each different test case. Failures in unexpected locations within the code are therefore included in the testing process, helping us to ensure an overall greater quality.**
   2. Describe your experience writing the JUnit tests.
      1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate. **All failing test cases were built with an expectation that they return an exception that can potentially be handled within the code to prevent an outright crash or other irregularities at runtime. An example of such is:** Assertions.*assertThrows*(IllegalArgumentException.**class**, () -> { **new** Contact(**null**, "Evan", "Carter", "5556667777", "Cape Kobayashi Maru");}); **In this case, an Illegal Argument exception is thrown upon the failure of this code’s proper execution. While it is still a failure, it is an expected one, and therefore can be handled.**
      2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate. **The tests implemented for the Contact, Task, and Appointment classes all depended on instantiating test versions of those classes to work with. Constructors are a highly efficient method of building classes of this kind, that require being instantiated multiple times. A simple example of this is:** Contact testContact = **new** Contact("5", "Evan", "Carter", "5556667777", "Cape Kobayashi Maru"); **Being able to generate this** **object in a single line vastly cuts down on redundancy and allows for easier, faster testing.**
2. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details. **Using JUnit to test our code was a form of dynamic testing, as the code was executed and evaluated for irregularities during runtime. Each class was also subject to unit and integration testing by having their functionality evaluated in a vacuum and when interacting with other classes. In a way, having the code and JUnit tests assessed and given a grade is also a form of static testing as a review is conducted on the unexecuted code to check for faults or omissions.**
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details. **None of our testing practices were explicitly non-functional in nature. The requirements themselves were all functional statements and generating tests from them produce functional tests. While there are some exceptionally minor elements of efficiency considerations, such as the constructors discussed above, there was no thought given to usability or security.**
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations. **The static and dynamic testing methods used in our project are fundamental techniques that are commonly utilized in real development. Especially on a large project, constant peer review of work products is necessary to prevent errors from slipping through and achieving a high level of test coverage grows in importance as the scale of the codebase increases. There was no requirement for a user interface, but such a thing would absolutely be necessary for a real project and the testing of its functionality and performance would be required as well. The project would also be subject to security concerns, mandating best coding practices and a baseline level of data encryption. Though these were not involved in this project, it is valuable to consider what changes and types of testing would be necessitated by them.**
   2. Mindset
      1. 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. **As discussed before, one of the greatest examples of precautionary testing used in this project was creating failing test cases in addition to the successful ones. Instantiating a new Contact class required tests to ensure that it could succeed but also that it could fail and when it did fail, the outcome was as expected.**
      2. 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 reviewing the code, I made constant comparisons to documented best practices, Oracle’s Java documentation, and examples and suggestions provided in the course materials. I believe these helped to produce a mostly bias-free review, but I cannot say that with certainty. Left to create and test my own code on a real project, my bias would absolutely be a point of concern. Simply put, I am inexperienced in both aspects and would introduce and overlook many errors even with such resources. An easy example comes from my grading of the ContactService class, in which I twisted the requirements in my mind from simply adding a Contact object to a list to creating said object as well as adding it to a list. This kind of misconception could warp the final product significantly from what is expected.**
      3. 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. **A great deal of software that exists today handles exceptionally important systems that people depend on. Failure in these systems is often not tolerable, potentially causing loss of reputation or investment or even creating dangerous situations that could lead to injuries or deaths. A premiere example of this is the case of a lawsuit against Cener Health Services as discussed in the article “Electronic Health Record Goes on Trial: Negligent Design That Resulted in Brain Damage” (Harrington, 2024). In this case, a patient suffered serious physical complications due not necessarily to faulty programming, but to user interface design lacking in appropriate information. This illustrates the value of extensively testing all aspects of a software project’s design, not simply the functional requirements. Avoiding technical debt and other such issues requires a multi-faceted approach to testing that applies a variety of testing techniques and works to the strengths of the team members involved.**

# Works Cited

Harrington, L. (2024). Electronic Health Record Goes on Trial: Negligent Design That Resulted in Brain Damage. *AACN Advanced Critical Care*, 10-13.