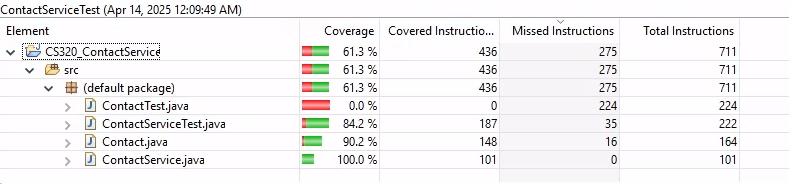
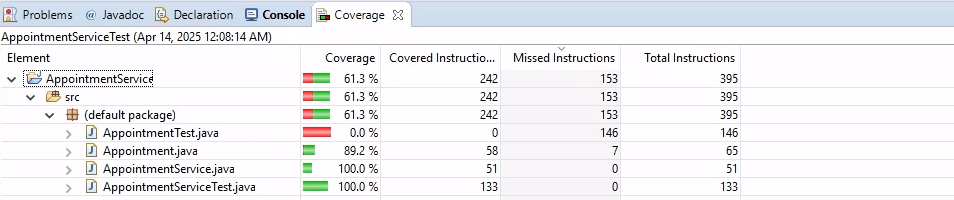
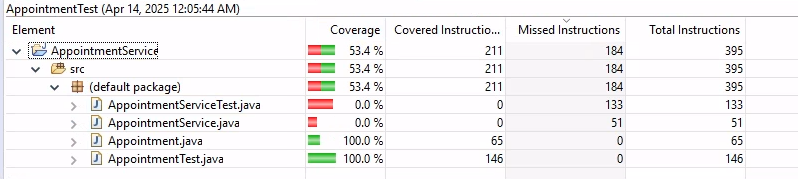
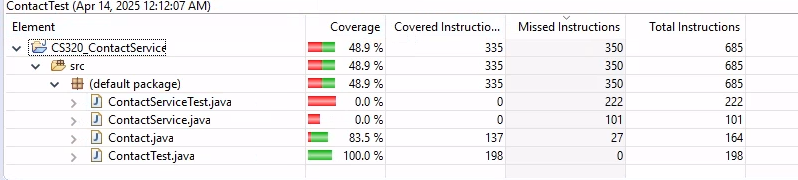
CS320 Software Testing, Automation QA  
Module 7 – Project 2  
Angelo Luo  
Travis Williamson

4/25/2025

My approach was closely aligned to the software requirements. Using the requirements I created a series of unit tests to test for as many identifiable points of failure in the application as possible and then developed the application around the user requirements, ensuring that each new section of code passed its tests prior to moving on to the next component.

Testing was thorough, with each test suite covering between 85 to 100% of its section of the application. In all three instances, the Service test suite also covers the majority of the functionality of its associated objects functionality. Furthermore, all tests were closely aligned with the software requirements identified in the design documentation and



I ensured that my code was technically sound through a process of writing small sections of code, ensuring successful compilation, testing that section through the test suite, and reworking / refactoring as needed for each section to ensure that the code was both performant and met the software requirements. As each section was completed, more of the test suite was brought to bear against the application, including additional test code at various points of the application, in order to ensure that it was not only meeting the requirements but was doing so in the expected manner. 

In order to ensure that the application was as efficient as possible, each test relied on locally created objects and variables, then released them for the garbage collector as soon as the test was complete.

Manual testing: During development multiple break points, print statements, and additional test code were used to verify that the applications output met expectations  
Automated testing: A series of 6 test suites were used to test each component of the application against the software requirements  
Functional testing: The majority of testing was against the functional requirements of the software.  
Unit testing: JUnit tests were created as part of automated testing, in order to test the code and ensure that all components were working correctly together.  
Integration testing: Each of the three separate major components were tested to ensure that their attendant Service class was able to create and manage their associated Objects following the outlined software requirements.  
Regression testing: As the software was developed, any refactor of the code brought an additional round of testing. Most commonly this testing was in the form of manual testing using line breaks and print statements. Finally after each refactor was completed, having passed manual testing, the automated suite(s) were run again to ensure that the code continued to behave as required.  
BVT: In each test suite, tests were included to ensure that the upper and lower bounds for each variable/parameter passed and that values outside that range did not.

System testing: Each Service test suite includes test coverage both for that Service and its attendant object type, ultimately the three Services do not currently interact with each other.

Ultimately, these tests did not include Non-functional testing, including performance testing, usability testing, and security testing. The application is currently very small and has limited functionality, identifying performance bottlenecks was not a priority during its development.

While efforts were made to ensure that the variables, methods, and classes access level was appropriate for the expected level of interaction with the user, no tests covered software security. Finally, no usability testing was included, as currently the software does not have any form of direct user interaction.  
Each of the software testing techniques employed has an important role to play in the development and testing of any piece of software. Manual testing was used to test small sections of the code for functionality, at a time when the larger section was not ready for the test suite. Automated / Unit testing enabled the rapid testing of the larger sections of the software, without having to introduce large amounts of one off test code inside the application. Regression testing is important to ensure that the codebase remains functional during any refactoring of the code and ensure that changes do not adversely impact the application.

During development I employed caution by ensuring that the test suite(s) were as thorough as possible, covered as many potential edge cases as could be identified, and verified that the tests themselves were accurate to the expected behavior of the application.

I can imagine that the level of bias a developer might have in testing their own code could impact their ability to remain impartial and could result in a case of ‘but it works on my machine.’ Testing should ultimately be carried out at a minimum on a variety of setups, ensuring that no platform specific features are used without consideration for future deployments. Further, a developer may know exactly what their tests should see based on their code and potentially miss edge cases or weak areas in their program.

Remaining disciplined and avoiding cutting corners during development helps to ensure that the program remains maintainable, within scope, and prevents unexpected behaviors from emerging later in the applications life cycle. Avoidance of technical debt, where possible, is a key component of my programming strategy. One of the most important principles introduced through my experience in game programming is a focus on ensuring that my code follows the idea of maintaining loose coupling and tight cohesion. As much as possible, one section of an application should not be reliant on the exact implementation of another portion of the software.