July Wellman

CS320: Software Test Automation & QA

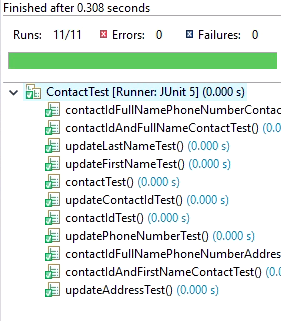
Southern New Hampshire University

December 10, 2023

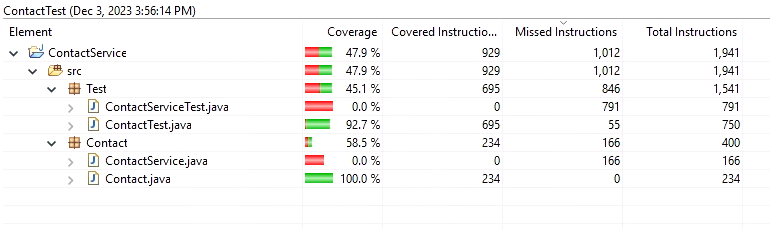
**Grand Strand Systems Report**

**Summary**

For Grand Strand Systems, I developed unit tests for the contact, task, and appointment services. My project was not aligned to the software requirements up until the last submission where I had to bundle all three and send it off as a complete package. That was the point where I was able to get a better understanding of unit testing and how coverage can be tested to ensure good results. To ensure the tests were effective I used Junit tests and coverage testing to ensure that the code executes correctly. I ran a coverage test on the contact, contact service, task, task service, appointment, and appointment service. Each test I ran to meet the requirements that all tests passed and had a coverage of at least 80%.



First, we tested to ensure that the functions of the code will work like updating a person’s first or last name. In the above image, you see the result of the JUnit test that was run to test the many functions of the contact class. All 11 tests ran successfully with no errors or failures, which is what we want to see when running a JUnit test.



The above is the result of a coverage test for the Contact Test. You can see that the contact.java tested at 100.0% coverage and the ContactTest.java tested at 92.7% which meets our 80% minimum threshold to meet system requirements. I ensured that my code was efficient and technically sound by running a coverage test often when I added or made changes to the code. Coverage testing helped me pinpoint areas in the code that needed attention. Many of times, it was due to a syntax error or piece of code that I failed to include. Using coverage testing and individual Junit testing helped identify problem areas cutting down on the time that I have to spend looking for errors in the code. I also learned that there is a difference between @Test Each and @Test Before Each which was very insightful.

**Reflection**

Software testing techniques that I used in this project were unit testing, integration testing, and system testing. I used unit testing to ensure that all the functions in a class were working properly. Integration testing was used to ensure that classes like the task and task service were working properly together. System testing was used to ensure that everything ran as intended. Software testing techniques that I did not use were performance, load, and acceptance testing. The project did not have a requirement to test for any of these so these types of testing were not done however, they do prove their uses in different projects that would require this kind of performance testing.

The mindset that I adopted for working on this project was to understand the system requirements and build a program around these requirements. Acting as a software tester, I always employ caution. It is very easy to change one character and cause a ripple effect on the rest of the code without even knowing, therefore caution should always be exercised. When reviewing the code, I made a conscious effort to limit bias by focusing solely on the technical aspects and adherence to best practices, rather than any personal opinions or preconceived notions. Being disciplined in my commitment to quality as a software engineer professional is a key factor in determining the success of a software project. It is easy to lose focus and let quality slip. It is important not to cut corners because doing so can lead to technical debt. I experienced it firsthand with this project and learned the value to writing clean code from the start and not cutting corners to avoid technical debt at the end of the project.