Trent Hesler

CS320-H7011

12/15/23

**Assignment 7-2: Project Two**

For testing, I followed the “methodical strategies” described in our textbook (Hambling, 2019, p. 167). To determine which tests to perform, I would first review the rubric for the given assignment, taking note of the listed requirements. From there, I designed a JUnit test to correspond to each of the graded requirements from the rubric. JUnit testing is a form of static testing involving test scripts.

Static testing uses source code, requirements documents, source code review, and automated test scripts to assess programs. All testing performed on this project falls into the category of static testing. Project One was subjected to manual code review, manual review of adherence to requirements documents, and automated test scripts. These tests ensure that the program compiles and runs properly and meets the requirements set out in the rubric.

Most other forms of testing would fall under the dynamic testing category. Dynamic testing generally involves performance and system resource usage testing, security or vulnerability testing, and verification of the quality of application output. Dynamic testing tends to be more abstract and thus more time-consuming. As this was simply a school assignment, this level of effort and scrutiny were not warranted, and the time requirements would have been incompatible with the course timetable.

During testing, for any rubric requirements that I felt were somewhat ambiguous or which were multifaceted, I created multiple tests to ensure coverage and client satisfaction (aka favorable grading from my professor). A clear example of this can be seen in the ContactService.java file from Project One. After reading the rubric, I was unsure whether the contact IDs were meant to be provided from an external source and checked for uniqueness or whether they were to be generated by the ContactService class’s addContact function. In the end, to ensure compliance with the software requirements, I chose to create an overloaded addContact function to handle both scenarios.[[1]](#footnote-2) This also required the use of two separate JUnit tests to address both function variants.[[2]](#footnote-3)

To ensure adequate test coverage, I ran the full battery of JUnit tests for each object class and then inspected the coverage report Eclipse generated for each test being performed and each method contained in the system under test. If the reported coverage for any of these was inadequate, I then made use of the code highlighting feature in Eclipse. This feature highlighted, in red, the lines of code that were not executed during a test, and highlighted, in green, the ones that were. If significant portions of code were not executed during testing, I would either refactor the test code to make use of those portions or refactor the object method’s code to remove redundant or unnecessary instructions. This also improved the overall efficiency of the code by removing unused code.

To cite a specific example, the version of the Appointment Service I submitted in Module 5 didn’t include any tests covering the Appointment class’s getter methods.[[3]](#footnote-4) This oversight was corrected in my submission for Project One.[[4]](#footnote-5)

To ensure the technical soundness of my code, I was sure to include JUnit tests that addressed scenarios where the methods would be passed or attempt to access invalid data. A clear example of this is present in TaskTest.java from Project One. In this battery of tests, I created JUnit tests that addressed all invalid arguments that could be passed to Task’s constructor class[[5]](#footnote-6), as well as tests confirming that a Task object instance created with valid attributes could not have one of those attributes changed to something invalid.[[6]](#footnote-7)

At my previous university, a computer science professor once stated that a good programmer should try to “imagine every possible way someone could break your program by providing invalid or nonsensical input.” This is advice I have taken to heart, but it has proved to be a bit of a mixed blessing. In my quest to safeguard against invalid input, I sometimes wind up creating redundant and inefficient code. For example, in Module 5, the addAppointment method in my AppointmentService class included guard clauses protecting against null pointer exceptions.[[7]](#footnote-8) I had assumed that these exceptions would have been thrown by attempting to add an Appointment to an AppointmentList that had been deleted.However, my testing for Project One lead me to discover that these guard clauses were unnecessary. They were removed from the AppointmentService class included in Project One, making the code leaner and more efficient through the removal of unnecessary logic.[[8]](#footnote-9) When analyzing the results of my JUnit tests, I would note which lines did not execute and ask myself whether it was because they were intended to address a scenario that I was not currently testing for or whether they were just redundant. If I determined it was the latter, I would remove the code to increase overall program efficiency.

I would describe my approach to creating this program as cautious. For each method, I first developed the core functionality, then backtracked to add in sanity checks and guard clauses for all of the ways I could imagine things going wrong. I also used an iterative approach. For example, in Module 5, I started out by creating a bare bones version of only the Appointment class. I then added the required methods one at a time, using printed output statements to verify proper functioning. I would ensure that each method compiled and ran correctly before moving on to the next. I think this approach is best because, by using an iterative approach, you minimize the amount that can go wrong at any given time and lessen the amount of code that must be combed through to resolve issues.

Try as one might to be objective and critical of their own work, however, one can never be completely unbiased. This is why it is important to seek other opinions and perspectives. Someone with a different point of view may pick up on something you would never have noticed. For this reason, I believe it is best to have separate parties developing software and performing tests of said software. This also has the benefit of distributing the workload so that no one person or team is responsible for doing everything. Again, because this was a university course, the circumstances obviously didn’t allow for this.

Lastly, I believe that it is important to be disciplined and thorough when creating software. I myself take pride in producing as quality of work as possible. Besides this, however, developing robust, efficient, and reliable software is crucial to gaining and maintaining business in the software development (and any other) industry. If clients can’t trust the software you or your company produces to meet their needs, they will likely take their business elsewhere and certainly won’t recommend you to their associates. So, not only is developing quality software crucial to the satisfaction of your client, it should also be in the best interests of you, professionally.

**References**

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition* (pp. 167-168). BCS Learning & Development Limited.

1. 3-2 Milestone: Contact Service, ContactService.java, Lines 77-115 [↑](#footnote-ref-2)
2. 3-2 Milestone: Contact Service, ContactServiceTest.java, Lines 21-45 [↑](#footnote-ref-3)
3. 5-1 Milestone: Appointment Service, AppointmentTest.java [↑](#footnote-ref-4)
4. 6-1 Project One Submission, AppointmentTest.java, Lines 128-159 [↑](#footnote-ref-5)
5. 6-1 Project One Submission, TaskTest.java, Lines 23-81 [↑](#footnote-ref-6)
6. 6-1 Project One Submission, TaskTest.java, Lines 138-180 [↑](#footnote-ref-7)
7. 5-1 Milestone: Appointment Service, AppointmentService.java, Lines 26-66 [↑](#footnote-ref-8)
8. 6-1 Project One Submission, AppointmentService.java, Lines 26-62 [↑](#footnote-ref-9)