CS320: Software Test Automation and QA

Professor Rangitsch

Tris Bonomo

February 20, 2022

**Summary and Reflections Report**

**Summary**

In my testing approach I focused exclusively on meeting the stated software requirements. According to the design specifications, I was required to create add(), update(), and delete() methods, but I created an additional retrieve() method. Since update() and delete() need to access the contacts/task data structure using a unique ID, I needed to have code to handle the search functionality. For the sake of modularity, that meant creating a method that both methods could call to complete their unique processes. Though retrieve() is an essential part of my code functionality, it is not the only possible solution for this project. Since retrieve() is not explicitly required by the code documentation, I excluded it from my test cases, and ignored lower coverage percentages in code areas related to it. I only focused on achieving 100% test coverage for code functionality that was expressly stated in the prompt. It did not make sense to over engineer tests for code that could potentially be stripped out in the next phase of project development.

Before this class, I would typically write in-line code to test my code solutions. Using hand-keyed inputs, I would run and debug my code to find errors. I used a combination of my former process along with testing on the Contacts assignment but only I used JUNIT testing for Tasks. Following standard coding practices and debugging my testing code, I was able to identify an error in my deleteTask() method for TaskServices.java. I initially forgot to put a return statement inside if statement nested in the foreach, which caused my test for the method to throw an error. I cannot provide code examples of this particular issue because I corrected the code to remove the error.

Due to the simplicity of my current code and the fact that the requirements did not state any efficiency requirements, I did not prioritize it while writing my program. Instead, I focused on writing elegant code solutions. For instance, I used a different approach for the deleteTask() method in TaskServices,than the one I used for ContactServices.java. The method went from two lines to four, which is probably less efficient. However, my deleteTask() method has handling to ensure that the passed in ID isn’t null before calling remove. Despite the potential efficiency decrease, I believe it is a better code solution.

**Reflection**

For each of the project milestones, I primarily used specification-based techniques like equivalence partitioning, boundary value analysis, and decision table testing, but also used error guessing, which is an experienced-based technique. Specification-based testing test cases are developed based on the project’s design documentation, while experienced-based techniques rely on a developer’s programming and testing knowledge. When the milestone documents specified things like variable types, character lengths, and acceptable outputs, I used that information to guide my code design and test cases. If I was given a maximum string length, that helped to set the values for equivalence portioning and boundary values, while information about unacceptable outputs, like ‘null’, helped me assess my decision table set up. When it was time to test my code, I used that same information to set the parameters for each test. My experience-based techniques were developed as I worked through each consecutive milestone. A test to check that a string is no longer than X characters long will be the same for a string that is ten characters or a string that is 50 characters, so I reused several test structures week to week.

Structure-based techniques are developed based on the overall structure of a codebase, so I did not use them for any of the milestones. The project guidelines did not dictate a particular program structure, so I was able to develop my code however I saw fit. Though my programs do use conditional statements and loops, structures according to the book, I did not use them in any of my tests.

Developing code and tests based on project requirements and experience is only logical: software is developed to fulfill a specific expectation, and developers should constantly reference their experience to meet experience in a timely fashion. However, the benefits of structure-based testing are more abstract. In structure-based testing, tools like pseudocode and flowcharts are used to demystify code structures by translating them into human language or charts. The primary benefit of this sort of testing is that can be used to explain technical problems to non-technical business resources.

In acting as a software tester, I employed caution by ensuring my test coverage accounted for code interactions. In lines 17-49 of my AppointmentTest.java file, I used three separate tests to assess my setAppointmentDate() and getAppointmentDate() methods. Since the getAppointmentDate() method retrieves values assigned using setAppointmentDate(), I knew I had to test the two together. If I had tested my get/set separately, my code could have passed all my tests without assessing their primary functionality: retrieving a value that has been set. My approach allowed me to achieve full test coverage while testing both methods simultaneously.

Unfortunately, I cannot claim I attempted to limit bias during my code review. I created both the code base and test cases, so I am the least qualified to assess the efficacy of either. I do not think code review was biased, but I would not swear by it. Proofreading my work has never been my strong suit, so I prefer to leave double-checking to others. When I discovered coverage deficiencies in my tests, I defaulted to the software and adjusted, and that's as close as I got to limiting my biases.

Cutting corners and producing subpar work is detrimental no matter the vocation: there is no value in pretending software engineering is unique. Doing something the right way takes less time than cleaning up the mess when things go awry, so it pays to do them correctly the first time. As a practitioner in the field, I plan to do the work assigned to me to the best of my ability and stay open to learning new things. It is not a surefire way of avoiding technical debt, but I'm confident it will count for something.