John Elbogen

CS 320

2/21/2021

**7-2 Project 2**

My testing was fully aligned with the software requirements. The Junit tests I performed covered, Contact Service, Task Service, and Appointment Service requirements. I did this using an exception throw test for characters one over the length limits. For example, the testContactIdTooLong() function, which checks creating a new contact with an id over the 10-character limit. This checks our ID length, and the same Junit format was used for all repeating instances of id length tests. Next, I checked my service methods by adding an object with a unique ID using the add() functions. The test checks that an object is added, and the list containing the object becomes populated. If this list size is zero than the add method does not work correctly. The process was similar in the delete methods, but instead I call the add function followed by a delete function and test for a list size of zero confirming the added object was deleted. This works because I already confirmed a Junit test on the add method. If that passes, I can use it to populate and remove from a list which will fully test the remove method. Before each check I cleared out the existing list to make sure each test was independent, using the clear() function.

The quality of the Junit tests was effective because they had high coverage. Nearly all lines were tested. Some lines that would specifically throw an error in my Junit test were not covered, but that indicates that the exception was thrown. A line which was not tested was in my conditional statements that check for null entries. This coverage may be unnecessary as it will function the same in all statements, compared to the length tests, that often change values.

I ensured my lines were technically sound by making sure each line used proper conventions and contained no errors. For example, the line, if (id.length() > 10 || id == null) { throw new IllegalArgumentException(“invalid ID”) }. In this line I ensured it was technically sound by using non-functional testing and reading the code. I translated it into how it should function. It reads as, if the length of the id is greater than 10, or it is null, then throw an exception. Using this non-functional testing, I can assure that the logic makes sense. Another line that illustrates this idea is *assertTrue*(task.getId().equals("id")); By applying the same idea as above, I considered the assertTrue function, and know that it should return a true value to pass the Junit test. By using task.getId() I should return the current id string of the task. This was set to “id”. Then by convention, .equals(“id”) should check that this is equal to the string “id”. Logically, id is equal to id. This should return true from the function and give a passing test.

To assure efficient code, I made sure that I wrote reusable code, that could be adjusted for each test. An example of efficient code is the line: *assertNotEquals*(TaskService.*tasks*.size(), 0); This line shows using the Junit assertNotEquals function. To have efficient integration, I used this method versus manually checking using != and assertTrue. This is also more efficient because it is reusable. Given the code must be edited in later iterations, the line assertNotEquals will specify that the two elements being compared must not equal each other. This method can accept multiple data types, making it more flexible/adjustable.

I utilized the following testing techniques in this project. Unit testing, Boundary Value Analysis, State Transition testing, and Decision Table Based Testing. Beginning with unit testing, this is simply taking each method and testing that they function on their own. An example of this was from my Appointment Service program, which tested the creation of an appointment. Next, boundary value analysis testing was used. To do this I would test a method manually inputting boundary values. For example, the id could not be longer than 10 characters long. I would create an id with 11 characters and test that an error is thrown. This is testing the boundary values within my program.

The next testing utilized was state transition testing. In this test I would perform multiple methods, to test a function at a specific state. An example would be testing the delete function. Since my delete function would only work when the program is in a state where an object exists in the list, I needed to reach that state and then call the remove function, before testing my result. Finally, decision table-based testing was used. This was implemented minorly in some areas. Within each test, I checked for multiple possible combinations of a condition. For example, what if I entered an id which was too long, while all other information was correct, and the next test I entered a description that was too long? By testing each case I could confirm the program was running as intended and the cause-effect relationship functioned properly while getting full coverage.

Some of the following testing techniques were not used: System testing, Performance testing, and Compatibility testing. System testing is simply testing the software in its entirety. I did not explicitly test the entire program, because the code required individual unit testing. Performance testing would have involved multiple types of stress tests, and endurance checks. For example, I would check how the methods would handle 1,000 inputs, versus one input. This was not necessary in my code requirements. Finally, I did not use compatibility testing. The program specifications do not need to check how my program would work in a different environment. An example of this would have been testing my program on Mac OS versus Windows OS.

Each of these techniques have their place in software development projects. Unit testing is important especially on large scale projects. It helps confirm each unit within a project follows correct logic and functions as intended. Boundary value analysis is critical when dealing with exact values or data. Errors are often found at the boundary values in programs, and this is a preventative test that saves time in the long run. Spotting a boundary value late in a project can be difficult without adequate testing. State transition testing finds value in any program that deals with different outcomes based on the state. If a program only functions in one state, then there is little value to this type of testing. Decision table-based testing allows for large coverage. In any complex program, decisions table based can help organize and understand what needs to be tested for maximum coverage. This is practical in almost all programs. System testing assists towards testing milestones within software design.

While unit testing is great for catching errors, without testing the program in its entirety, the cohesion of the units cannot be tested. This is always necessary and completed at some point in the SDLC. Performance testing has an important position in programs which push hardware and software. The other case where this testing is extremely crucial is where a program must service many users. A system may break or become painfully slow under the stress of millions of requests, and function perfectly with thousands of requests. The only way to know this is with performance testing. Compatibility is paramount in today’s programming environment. A product usually requires mobile and multiple OS compatibilities to become successful. Compatibility testing allows the developer to know that a program will function seamlessly regardless of platform (iphone, android, PC, Mac, etc.).

During this project I adopted the mindset of the user. Many times, I considered what would happen if I (as the user) did x event? There were many times I had to be cautious in my mindset of my own knowledge using a program. It was important for me to consider the complexities and interrelationships of the code I was testing, because it allowed me to be much more efficient in my testing and choose correct approaches. An example of this would be testing creating a new Contact object. In the rest of my code, I would want to create an Appointment, and Task object. Through this consideration, I decided to write a test object function, that can be reapplied in each situation, with minor adjustments to the information that the object receives. For testContact(), I created a new contact with an id, fist name, last name, phone number, and address. Using the same layout and code I was able to edit the input data for testAppointment() by passing in an id, date, and description. This made my code much simpler and more reusable.

To limit my bias throughout the review of code, I checked that all cases were considered. Rather than assume the user will follow the intention of the program, I prepared exceptions. For example, throw new IllegalArgumentException(“invalid ID”). This line throws an exception if my specific condition is not fulfilled. By coding in this way, it covers all cases outside of what is acceptable, including any assumptions and bias I may have. Being disciplined in commitment to quality is crucial. Cutting corners can lead to poorly tested software. One misstep on a project can lead to complete failure or wasted hours of debugging. User’s information, safety, and livelihood could be at stake. I plan to avoid technical debt by developing a case for myself in my work. If I cover all foreseeable bases and put full attention into what I am working on, I can be assured that I have done everything I can do, with proof. Also, I will use my resources to the fullest to assure my work is well done. This might mean asking someone with more knowledge to check for inadequacies!

**Citations**

Tutorial Using JUnit. (n.d.). Retrieved January 31, 2021, from https://learn.snhu.edu/content/enforced/636498-CS-320-T3229-OL-TRAD-UG.21EW3/course\_documents/CS%20320%20Module%20Four%20Tutorial%20Using%20JUnit.pdf?\_&d2lSessionVal=IwM6cAewn9frqxb53vnJ51ZLs&ou=636498

Software testing methodologies. (n.d.). Retrieved February 08, 2021, from https://smartbear.com/learn/automated-testing/software-testing-methodologies/

Software testing techniques with test case design examples. (n.d.). Retrieved February 08, 2021, from https://www.guru99.com/software-testing-techniques.html