Ashley Sapunarich

CS320 Software Test, Automation QA

12/14/2024

**Project Two**

When it comes to the Task Service Project, I made sure my testing matched the software requirements by not only testing for cases of null, but also length issues as well. I ensured that I had tests for situations such as Task ID, Name, and Description too long, as well as null. This aligned to the software requirements because they stated such: The task object shall have a required unique task ID String that cannot be longer than 10 characters. The task ID shall not be null and shall not be updatable. The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null. The task object shall have a required description String field that cannot be longer than 50 characters. The description field shall not be null. My tests were designed to align with the software requirements to make sure that all requirements were met. In regards to the JUnit tests on the task service project, I spent hours trying to ensure that the coverage percentage was as high as I could possibly make it. I also made sure that I had as much green throughout my tests, and I did not have any spots declaring missing branches.

I ensured my code was technically sound through the use of the IDE itself. Eclipse is an amazing tool in that it guides you when it can detect what you are meaning to code, but not what you actually coded. There were a couple of times when my code was not quite right, it has been a while since I have coded so it is bound to happen. Eclipse helped guide my direction of coding, as I was typing in commands such as Assert.assertEquals("This is acceptable", task.getDescription()); - I originally just put getDescription() and did not include task. which caused an error. Clicking on the error itself, Eclipse offers you with a set of potential solutions. These helped guide me in the correct direction as I am still a less confident than I would like in my coding abilities. Additionally, researching what I needed or what trying to think of helped a lot as well. I forgot how constructors worked, and needed to go back into my old books to read up on them again. This ensured I was using them correctly at the very least.

As far as my codes being efficient, I attempted to do so by including multiple arguments in the same line of code. A good example of this is with Task, as here I put all arguments under the same line, from not being null to not being over a certain number of characters. “public Task(String taskId, String name, String description) { if(taskId == null || taskId.length() > 10) throw new IllegalArgumentException("Invalid Task ID"); if(name == null || name.length() > 20) throw new IllegalArgumentException("Invalid Name"); if(description == null || description.length() > 50) throw new IllegalArgumentException("Invalid Description");” This code allowed me to place all parts of a task into one area with all requirements as well. I believe that this made my code more efficient than my previous contact class, where I did everything individually.

When it comes to the milestones from Modules Three, Four, and Five, there has been a lot of boundary value analysis and equivalence partitions. There has also been a lot of error guessing in my case as it has been a while since I have coded, and it shows in my work. Boundary value analysis and equivalence partitions are testing techniques that are generally seen used together. In the case of all of the milestones, we have seen them work together in that we have a set of boundaries in the character counts of the strings. An example of this would be that the IDs cannot be null and cannot be more than 10 characters long. This gives us our boundary of 1-9 as 0 is considered null. Error guessing is not a technique that should be used often, but in my case the areas of my code that I was least confident in writing were the first parts of my code that I put through a test. This at least allowed me to be sure that I was on the right track, or know that I needed to go back and more thoroughly rethink my code.

Statement testing and coverage can also be seen across the milestones. Each milestone had a set of statements that the class and service were needed to have. An example of this is “The appointment service shall be able to delete appointments per appointment ID." This tells us exactly what we need to test just based on the statement given. In this case, we must test that the appointment service class is able to delete appointments based on the given appointment ID. Coverage is also seen here, as in the Junit testing, we were putting out focus into getting as close to 100% as possible. I was not successful in this at all in the first milestone, but did better in the other two. I still struggled with finding the gaps in my coverage, but was at least knowing that I was in the right direction going from a 30% to an 80%.

There were some testing techniques that were not used in these milestones. Decision testing is one of them, as it focuses on loops in code which I did not have. Decision testing ensures that both cases of true and false are tested to ensure complete coverage. Simplified control flow graphs also are another testing technique, which share the format used for flow charts. They exist to identify control structures but not the detail of code. This can also help in visualizing the placement of a loop in a code. Additionally, we did not use exploratory testing which combines experienced testers with a structured approach to testing. We also did not use checklist-based testing which is based on high-level checklists, which can be drawn from many different sources.

When it comes to the testing techniques that we did use, I believe that all of them can come in handy in future projects. I also believe that the actually concept of testing is important for all people interested in coding to learn, as it helps see a different light to what I am coding. Error guessing is far from the best testing technique in my opinion, but it was valuable as someone who is not confident in their coding after a while of not doing it. I was able to pick out where I was the least confident, and most of the time I was correct that it did not pass the test because there was an error in the code. Boundary values are important because they serve as almost a checklist for testing a project. If something must not be null or over 20, you automatically two tests you must complete just on that statement alone. I believe all of these testing techniques will not only serve as useful in the future, but would have helped me greatly in the past as well.

I employed caution in every aspect of this project. I was extra cautious in regards to my code especially. As stated previously, I was relying on the tips and errors that Eclipse IDE found. I was also using the help of websites like Stack Overflow to get answers involving code I was trying to get it work. A good example of this was the date, I was struggling to get the date to work correctly in the appointment section. I was getting frustrated and I used the help of these websites to get better explanations on how to properly use this. I appreciated the interrelationships between all of the code as it helped me realize the pattern. I was able to easily go back to my first code, contact, and add in the many tests I was missing.

I tried to limit my bias by walking away for a bit after coding. This helped give me a fresh set of eyes during the testing phase, and look at my code in a different perspective. There can be bias with one’s own code, as if they believe it is perfect with no requirements missing. I fell bias to my own ego with the coding of the task and task service, but upon testing I found that my own bias had gotten ahead of me. My testing was only 45% at first, regardless of how well I assumed I did. It is important to be disciplined in writing and testing code. As we found in a discussion, errors in code have led to fatalities and serious injuries. In practice, I aim to always do my best with code, as well as learn from past mistakes. My goal is to continue to utilize the tools that I have learned to keep making my code more secure and meet requirements.