Daniel Behmer

8/9/2022

CS-320

Prof. Kalinowski

Project 2

Throughout the course, we conducted a series of milestones that would involve Junit tests. Contact, Appointment, and Task all had their own service, as well as tests for both. For example, we would have contact.java contactservice.java, contacttest.java, and contactservicetest.java. Each of these had their own requirements. Let’s take a look at the appointment requirements for example. The appointment class requirements are the appointment object shall have a required unique ID string that cannot exceed 10 characters, and the ID shall not be null or updateable. The appointment object shall have the required Date field, and the Date cannot be in the past or be null. Lastly, the appointment object shall have a required description that cannot be longer than 50 characters or be null. The appointment service requirements are the appointment service shall be able to add appointments based on a unique appointment ID and the appointment service shall have the ability to delete appointments per appointment ID.

To follow these requirements, I had structured my code to follow the requirements and ensure that we were able to hit all points in our test cases. Here is some code that deleted an appointment:

@Test

**void** test02\_delete\_appointment() {

Appointment app = **new** Appointment("App01", **new** ~~Date~~(2022, 08, 05),

"Dentist appointment");

Appointment app1 = **new** Appointment("App001", **new** ~~Date~~(2022, 08, 05),

“Surgery appointment");

*appService*.addAppointment("App001", **new** ~~Date~~(2022, 08, 05),

"Dentist appointment");

*appService*.addAppointment("App001", **new** ~~Date~~(2022, 08, 05),

"Surgery appointment");

Assert.*assertEquals*(app.getDescription(), *appService*.getAppointMentList().get("App01").getDescription());

Assert.*assertEquals*(app1.getDescription(), *appService*.getAppointMentList().get("App001").getDescription());

*appService*.delete("App01");

Assert.*assertEquals*(1, *appService*.getAppointMentList().size());

The console output is “Deleted appointment ID: App01, Description: Dentist appointment”

Here, we deleted Appointment01 for the dentist and created a new appointment, App001, for surgery. We know the quality of the Junit tests is high because after the Junit test is finished, we look to see a green bar, with 0 errors, 0 failure, and all runs being completed. This essentially means that our Junit tests are without flaws and hits all requirements.

It is important that our Junit tests are both technically sound and efficient. Again, we can use the above example to showcase that it is both sound and efficient. By using the above example, we are testing the ability to delete a previous set appointment. Here, we delete App01, or the dentist appointment: *appService*.delete("App01");. From there, we can confirm that the appointment was deleted by checking the console with the output “Deleted appointment ID: App01, Description: Dentist appointment”. This indicates that our test case is functioning as should and is covering all requirements.

The software testing technique that I employed during this project was standard unit testing. This means that we are testing the individual components of the program. For example, AppointmentTest and AppointmentServiceTest. By utilizing unit testing, we are basically going down a list of requirements and using them as a checklist for how to structure our program. Some of the requirements are that the appointment shall have a unique required ID and the ID cannot be null, the object shall have a date that cannot be set to the past and cannot be null, and the appointment shall have a description that shall also not be null or more than 50 characters.

Utilizing the same example again of App01, we had a unique ID (App01), a date that was not set in the past (2022, 08, 05), and a description that was less than 50 characters (“Dentist appointment”).

One form of testing that was not used throughout this project was load testing. “Load testing is the process of putting simulated demand on software, an application or website in a way that tests or demonstrates it’s behavior under various conditions.” (LoadNinja). Again, using our example, if we were to inject the program with fabricated appointments, how long would it take, or how many appointments could be added until the program “breaks”.

It is crucial that software testing is done throughout the entire process of development. It is important that requirements are met as they offer two things for the product, functionality, and guidelines. By having these requirements, it can sometimes allow for a quicker and smoother development experience, without having to backtrack your work. By testing throughout the development phase, this eliminates this possibility almost entirely.

Due to the nature of the requirements of each case, caution was employed throughout the entire development process. Keeping in mind all of the requirements for each class, object, etc. was important for ensuring that our test would pass as they should. It was important to appreciate the complexity and the interrelationships of the code because in a way it helped with the development process. Some of the requirements were in line with others, so it allowed me to get an idea of how I wanted to structure and develop my code in a clean and efficient manner.

One way that I tried to limit bias in my own code review was to think of all of the ways that I would be able to “tear apart” my program. Basing the tests strictly on a pass/fail system for myself allowed to find errors within my program to fix, while also implementing the feedback received from previous milestones before the project was due. By following the guidelines given and implementing the feedback given as well, it helped me be disciplined as a software professional. It is extremely important to not cut corners in development. By cutting corners, it can lead to non-scalable, sloppy, hard-to-read code, and can cut a product’s life short. I plan to avoid technical debt as a practitioner in the field by following requirements, employing thorough testing procedures throughout the development phase, and by documenting my work. Ensuing that I apply these four qualities, I believe that I can produce clean, scalable, well-produced products.

Works Cited

LoadNinja. *Load Testing.* Loadninja.com. https://loadninja.com/load-testing/#:~:text=Load%20testing%20is%20the%20process,it's%20behavior%20under%20various%20conditions.