David Diaz

10/18/2025

Southern New Hampshire University

Project Two

When it came to testing the software requirements, I decided to test the variety of Object classes and Service classes. I went through the different Object classes, Contact, Task, and Appointments, and decided to test their initializations and the different property logic that each class has. For example, when testing the Contact class, we decide to test the proper instantiation of the object. We also decide to test the scenarios where errors should appear such as initializing a new object with an invalid ID, Name, or Address, or trying to set invalid properties on an already created object. Now while this is a small look at what coverage the tests have, I did decide to test every branch that the Object classes and Service classes could follow to create effective tests. Another example of effective coverage testing is when I tested the TaskService. Every function was tested in the service class was tested and verified in the TaskServiceTest class. I made sure that the CRUD operations that the service class needed performed properly and as intended. This involved testing the Add Task function, the Remove Task function, and the Update Task function. Within the Add Task function, I also tested the fact that a task cannot be added if the taskList already contains an object with a matching taskID. This avoids the ability to add duplicate objects based on the ID. Within these Service tests, there are also Task object functions that are additionally evaluated which adds another area that the object could be tested from. This shows how effective the tests I wrote were and it also adds additional layers of coverage to make sure that majority, if not all the project’s functions and objects were tested in their entirety.

The testing that I provided could be considered both effective and technically sound. The reason the tests could be considered technically sound was because of the various areas it covered while also keeping in mind potential edge cases such as adding a duplicate task as seen in line 21 and 22 of the TaskServiceTest file and verifying that errors would be thrown as intended as seen lines 16-18 of the TaskTest file. I also made sure to use the appropriate JUnit test assertions to verify that the best check was used as appropriate. While I explained previously why my tests could be considered effective, these explanations could be seen in lines 17-23 of the TaskServiceTest file.

While there are many testing techniques out there that can range from very basic tests to very complex testing, there are 2 that are very popular that I used in my testing, Unit Tests and Integration Tests. Unit Tests can be seen throughout the Object classes, this is because Unit Tests are meant to test individual components to ensure that they function correctly. I believe that this applies mostly to the Object classes because we are testing the building block of each object list and we need to make sure that the object itself is properly created and built out. One example of this can be seen when testing the Appointment object. In this object, there is the date property that must be set to be either today or any day after. It cannot be set to yesterday or any day before today without throwing an error. This had to be properly tested in lines 27-34 of the AppointmentTest file. This test function made sure that an isolated component of the class works as intended. Now when it comes to testing the different Service files, this is where Integration Tests come in. Integration Tests is a testing technique where multiple areas of the project are tested together to make sure that they can work together properly. This is why I believe that the Service Test classes work primarily as integration tests because they combine the Service layer with the Objects layer. This can be seen when adding an object to the object list as seen in lines 11-14 of the AppointmentServiceTest file.

While I only used Unit testing and Integration testing in my projects, there are other very popular testing techniques that I did not even consider. Two of those being Performance Testing and End-User Testing. I did not user Performance Testing because it was not specifically stated what speed the queries should be or what load the system should be under. This type of testing is primarily used to check for bottlenecks in your project and to make sure it is running as quickly and as efficiently as possible. I also did not do any End-User Testing because there was no external user to test the service to make sure it works as understood by the users. This type of testing is done to verify that end users get the results they were expecting when navigating and interacting with the different areas of the project. Now while I did not use the 2 testing techniques listed above, all these techniques have their place in a variety of projects. We use all these testing techniques and much more at my current place of employment because of the necessity to make sure our application is running smoothly and efficiently for all users.

When it comes to testing software, you want to account for every possible scenario that users can take instead of just the path that you hope users will take. For example, in the Object classes we placed check on the different properties to make sure that none could be null and that that they all had a certain range that they could not be bigger than. This can be seen in all of the object classes, but one special example is when setting the Date property of the Appointment object. In addition to making sure that the date cannot be null, the value must be equal to either today’s date or a future date and cannot be set to a previous date which can be seen on line 12 of the Appointment class. This example would throw an error if the value was either null or yesterday’s date for example. You must account for that and not just give full access to users hoping that they will not set the date to any day before today. Exercising caution is vitally important when it comes to software development.

One thing that is hard for anyone to do is removing bias when it comes to something you created. But there are situations, especially when it comes to testing software, where you need to remove that bias to make sure that something functions correctly and as required. One example of this bias is whenever I first submitted my Module Five assignment, I did not take into account testing to make sure that error would be thrown if any property was set to null on instantiation or update. This is something that I thought wouldn’t need to be tested because I knew that the code was written properly for this situation. That bias led me to not add that test which resulted in me losing points because of the fact. I now see that even if I know the logic is right, it is still important to cover every inch of the project to verify correctness for future development.

This also highlights the importance of making sure that you are committed to being disciplined when it comes to writing both your code and your tests. There are various reasons why one should not cut corners when creating your tests for your code, but the biggest one should be that if you test every area of your code then you will 99% catch any potential bugs before they can go out. Just like how we wrote about the errors in software when it came to this week’s discussion post, even the tiniest error can have massive consequences. I wrote about the errors in the Therac-25, a radiation machine that caused the injuries and deaths of 6 patients because of simple software bugs. This is why making sure one is disciplined in all areas of software is very important.

References

Leveson, N. (1995). *Medical Devices: The Therac-25*. University of Washington. <http://sunnyday.mit.edu/papers/therac.pdf>