Huan Ai

CS-320

10/5/2025

What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.

The primary technique I used for all three milestones was Unit Testing with JUnit. Basically, I wrote small, focused tests for each individual "unit" of my code—meaning each class and its methods.

For module three, I wrote tests to check if a contact ID was exactly 10 characters, if the first and last names were not too long, and if the phone number was exactly 10 digits. My tests would fail if I tried to create a contact with an 11-character name or a 9-digit phone number. I also tested the service itself by adding a contact, deleting it, and then trying to update its fields to make sure everything worked as expected.

For module four, the tests were very similar but focused on task properties. I made sure the task ID wasn't longer than 10 characters, the task name wasn't longer than 20 characters, and the description wasn't longer than 50 characters. The tests checked that the service could correctly add, find, and delete tasks based on their unique ID.

For module five, this was similar to the task service, but with an extra check for the date. I wrote tests to make sure you couldn't schedule an appointment in the past. This was a new kind of validation that the other services didn't need.

In all milestones, the characteristic of this testing was that it was automated. I didn't have to manually click buttons in an app; I just ran the test suite, and it would tell me in seconds if anything was broken.

What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.

I haven’t used integration testing and system testing in these milestones. For integration testing, since our services were simple and stored data in memory (like a simple list), we didn't need to test how they interacted with a real database or other complex systems. In a bigger project, we'd need integration tests to make sure all the different parts work together correctly. We didn’t use system testing either because we never tested the complete, fully assembled application. Our milestones were just the "backend" services. We didn't have a user interface (UI) to click through, so we couldn't test the entire system from a user's perspective.

For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

The techniques we used, unit testing, are the foundation of building reliable software. They are used constantly by professional developers. Whenever a programmer writes a new function, they'll often write a unit test for it right away. This is practical because it catches bugs early, when they are cheapest and easiest to fix.

One of the techniques we didn’t use is integration testing. Integration testing checks how different modules or components of a software system interact with each other. It’s done after unit testing and before system testing. It can help to verify that interfaces between modules work correctly, and detect bugs in communication between modules.

The other technique we didn’t use is system testing. System testing evaluates the entire software system as a whole. It’s the final testing phase before release and simulates real-world usage. Users will test both functional and non-functional aspects (e.g., performance, security) to make sure everything works as expected before release.