Jean Martinez

Professor Tuft

CS 320

**Project Two**

In software engineering, making sure that every part of a program works as expected is essential, and this is where unit testing becomes important. My project used Apache Maven, along with Surefire and Jacoco plugins, to test three main services: Contact, Appointment, and Task. These tools played a vital role in ensuring that each component of the software operated properly in different scenarios, which is fundamental for any successful software application.

The project focused on three main functionalities designed to improve operations and enhance user interactions within a software environment. The Contact service handled user contacts, managing tasks like adding new contacts, updating existing ones, and removing outdated information. It was important for this service to handle a large amount of data accurately to maintain user trust and protect data integrity. The Appointment service managed scheduling and overseeing appointments effectively to prevent overlaps and ensure timely reminders. Since timing is vital in managing appointments, its smooth operation was necessary. The Task service assisted users in tracking their duties and deadlines. It included features like setting priorities, updating task statuses, and sending alerts for upcoming deadlines, all of which helped boost productivity.

At the start of the project, I encountered quite a few technical challenges, including issues with setting up the Jacoco plugin and problems with importing the necessary libraries for testing. These difficulties demonstrated the complex process of setting up modern development environments and the importance of understanding the tools involved. To address these issues, I spent time researching and consulting the developer community, which not only solved the problems but also deepened my understanding of effective software development practices.

My testing approach was designed to thoroughly cover all aspects of the software. I aimed for comprehensive statement coverage and branch coverage, which required developing extensive tests that went beyond the basic functions. This included testing how the services would handle extreme or unusual conditions, making sure that new updates did not interfere with existing functions, and checking that the services could manage high volumes of data without slowing down. I achieved an 85% code coverage rate, which was crucial in determining the effectiveness of my tests and identifying areas where additional tests were needed to cover untested paths.

Debugging also played an important role in the project. It helped in pinpointing the causes of failures and provided deeper insights into the operational aspects of the code. This hands-on troubleshooting was invaluable in refining the functionality of the services and improving the overall quality of the software. Through these sessions, I learned the importance of patience and attention to detail, valuable qualities for any software developer.

Another significant learning area was the importance of thorough documentation. Throughout the project, I kept detailed records of the testing processes, outcomes, and changes made. This documentation was important not only for tracking the progress of the testing phases but also for future reference and for helping new developers understand the project. It ensured that anyone could understand why certain decisions were made and how specific challenges were addressed.

To make sure my code was reliable, I carefully checked all inputs and structured my methods to avoid errors. For example, in the TaskService class, the method addTask prevents common issues like handling empty or invalid tasks. Like this:

public void addTask(Task task) {

if (task == null || task.getName() is null || task.getName().isEmpty()) {

throw new IllegalArgumentException("Invalid task information");

}

tasks.put(task.getId(), task);

}

Efficiency was a key focus during the development of each service, especially in how resources were managed. By optimizing data handling and minimizing redundant processes, I ensured that the services operated efficiently, even under high loads. This was particularly evident in the Task service, where prioritization algorithms helped manage tasks effectively without performance lags.  
  
I focused on making the code fast and efficient from the start. For instance, the ContactService uses a hashmap to store and find contacts quickly, allowing nearly instant access in most cases. Here’s how the getContactById method works:

public Contact getContactById(String id) {

return contacts.get(id);

}

In terms of bias, I took steps to ensure that my testing approach was as unbiased as possible. This involved utilizing a variety of test cases, including those that simulated edge cases and unusual user behaviors. By incorporating feedback from peer reviews, I was able to identify and correct assumptions that might have influenced the test outcomes. I worked hard to avoid biases in my testing by using a wide range of test cases. This helps make sure the software works well in all sorts of situations, not just the ones I think are most likely. By testing thoroughly and in a disciplined way, I built a system that not only meets my needs but is also robust and reliable in unexpected situations.

My detailed testing confirmed the effectiveness of my approach, with all services showing more than 85% test coverage. Take the AppointmentServiceTest class, which thoroughly checks everything from simple tasks to more complex ones. Here’s an example:

@Test

public void testScheduleOverlap() {

AppointmentService service = new AppointmentService();

service.addAppointment(new Appointment("1", LocalDate.now(), "Doctor Visit"));

assertThrows(IllegalArgumentException.class, () -> {

service.addAppointment(new Appointment("2", LocalDate.now(), "Dentist Visit"));

});

}

Throughout this process, I learned much about different testing techniques and the language of Java in itself. Unit testing focused on verifying each function’s correctness, while integration testing ensured that different components worked well together. One of the biggest lessons from this project was the importance of a disciplined approach to testing, especially in consistently verifying the impact of new integrations on the system. It also taught me to be cautious, particularly when handling user inputs and data processing, to avoid potential security risks and system failures.

This unit testing project was more than just a task to ensure the software worked correctly; it was a profound learning experience that highlighted the importance of rigorous testing and careful planning in software development. The insights I gained from both the challenges and successes of this project are invaluable and will undoubtedly shape my approach to future projects, emphasizing the critical role of testing in building reliable software.

**References**

* Apache Maven. (n.d.). *Apache Software Foundation*. Retrieved from <https://maven.apache.org/>
* Surefire Plugin. (n.d.). *Apache Maven Surefire*. Retrieved from <https://maven.apache.org/surefire/maven-surefire-plugin/>
* JaCoCo. (n.d.). *JaCoCo Java Code Coverage Library*. Retrieved from <https://www.jacoco.org/jacoco/>
* Koskela, L. (2013). *Effective Unit Testing: A Guide for Java Developers*. Manning Publications.