Unit Testing Approach for Each Feature:  
Contact Service:

I focused on checking every detail of the Contact class such as the name length and making sure no details are missing. I also made sure the Contact Service can add, update, and remove contacts properly.

Alignment with Requirements: For example, I wrote a test to check that the first name is stored correctly:

assertEquals("Expected first name", contact.getFirstName());

This test checks that the first name matches exactly what we expect after creating a new contact.

Task Service:

My tests made sure that tasks couldn't be duplicated and that each task's details, like its name, met all the rules for length and content.

assertThrows(IllegalArgumentException.class, () -> new Task("longTaskID123", "Task", "Description"));

This test ensures that the task ID is not too long, matching the project's requirements.

Appointment Service:

I focused on making sure the appointment date was set correctly, ensuring no past dates could be used, and that appointments could be added, updated, and removed.

assertTrue(appointmentService.getAppointments().containsKey(appointment.getAppointmentId()));

This line checks that once an appointment is added, it can be found using its ID, which means it was successfully saved.

Writing JUnit Tests:

Writing these tests taught me to think about every possible situation the software might encounter. I learned to cover not just normal cases but also unusual ones to make sure the software can handle them.

Technical Soundness and Efficiency:

Example:

assertNotNull(new Contact("ID123", "John", "Doe", "1234567890", "1234 Street"));

This test confirms that when we create a new contact, none of the fields are missing or empty.

Example:

contactService.deleteContact("ID123");

assertNull(contactService.getContacts().get("ID123"));

This test checks if a contact is correctly removed from the system, which is done efficiently without extra steps.

Reflection

Testing Techniques Used:

Unit Testing: I checked each part of the software by itself, ensuring every single function works correctly on its own. This type of testing is very detailed and specific.

Integration Testing: (If used) This would mean checking how different parts of the software work together. For example, how the contact service interacts with the task service.

Other Techniques Not Used:

System Testing: This involves testing the entire software to see if it meets all the requirements. It’s important for checking the overall behavior of the system.

Acceptance Testing: Usually done by the end users to make sure the software is easy to use and does what it needs to do from a user's perspective.

Being careful was key, especially knowing that real people might use this software. For instance, I was very careful to check that all data types and values were correct to prevent errors. To avoid bias, particularly in reviewing my own work, I engaged in peer reviews and collaborated with others. Testing one's own code can lead to missed errors due to familiarity.

Sticking to high standards in writing and testing code is crucial. It prevents problems later on, which might take more time and resources to fix.

I plan to always follow strict coding guidelines and get my code reviewed by others. This helps avoid quick fixes that don’t solve problems thoroughly.

In this project, I learned the importance of careful testing and detailed checking. This experience will help me write better, more reliable software in the future.

APA Sources

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Microsoft Press.

Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing. Wiley.

These sources provide foundational insights into software testing practices and principles, reinforcing the concepts discussed in this report.