**Project 2**

1. **Summary**
2. Describe your unit testing approach for each of the three features.

ContactService: Test the functions of adding and deleting contacts, verify whether the contact fields are valid, and handle illegal input.

TaskService: Test the functions of adding and deleting tasks and verify the legality of task names and descriptions.

AppointmentService: Verify that appointments cannot be created for past dates, test ID uniqueness and exception handling.

* 1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The test methods are designed completely according to customer needs, such as, “The task object shall have a required unique task ID String that cannot be longer than 10 characters.”. This constraint is verified in the following code,

A computer code with text

Description automatically generated

At the same time, the subsequent code also verifies other customer requirements. For example, the name cannot exceed 20 and the description cannot exceed 50.

* 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

Regarding coverage, in TaskServiceTest.java, the four methods testAddTask(), testDeleteTask(), testUpdateTaskName(), and testUpdateTaskDescription() are used to test the validity of adding valid tasks, deleting duplicate tasks, and updating the name and description, basically covering all situations. In TaskTest.java, all possible non-compliance situations are also tested.

1. Describe your experience writing the JUnit tests.
   1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

In TaskTest.java,

A white background with black text

Description automatically generated

This corresponds to the logic of length description in Task.java.

A computer code with text

Description automatically generated

Another example,

A computer code with text

Description automatically generated

Corresponds to the name processing logic in Task.java

A close up of text

Description automatically generated

* 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

In Task.java,

A computer code on a white background

Description automatically generated

According to customer requirements, the ID length is limited to 10 and increases linearly, ensuring test efficiency and non-repetition.

For another example, in TaskServiceTest.java, taskList.get(0) is used to obtain tasks, avoiding repeated redundant code and excessive calculation.

1. **Reflection**
2. Testing Techniques
   1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

The most method used is Unit Testing, which mainly uses the JUnit framework for unit testing. Each service class (such as ContactService, TaskService, AppointmentService) has a corresponding test class created to verify its methods. The biggest feature of unit testing is that it tests the functions of a single module, with fast execution speed, obvious results, and no interference. There is also Boundary Value Testing, which verifies, for example, the name length limit by inputting the maximum and minimum values. The feature of this test is to verify whether the input logic is safe and identify input errors.

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

The method is not used: Integration Testing: Testing whether the interaction between multiple modules or services is normal. The main purpose is to ensure that there are no problems when different modules work together. System Testing: Testing the entire software to verify whether it meets customer requirements.

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

Unit testing is applicable to the initial stage of project development and the continuous development process to improve code quality. This is the most basic test. If unit testing is not completed, subsequent testing will not be possible. Boundary value testing is also applicable to the initial stage of development and the continuous development process to improve the rigor of input verification and improve code quality. Integration testing is used during the development process and the later stages to ensure that modules work properly. System testing is used in the acceptance stage of the entire project to verify whether the functions of the entire software meet the expected purpose.

1. Mindset
   1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

As a software tester, I am the final auditor of product quality. After testing, the product will directly to the customers. Testers should be very cautious in the final test. The complexity and interrelationships of the code are the basic logic. Only by understanding this logic I can understand the test results and solve the problems that arise in the test. For example, in ContactService, the IDs may be used directly for user input in AppointmentService.

* 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

Bias will always exist. For example, I think my own code is well written, but when I test my own code, I may ignore this part of the test because of my confidence. Logically, first, my own code is not perfect. Because I wrote it myself, it is difficult to find problems subjectively, so I need to prove it through testing. Secondly, even if my own code is perfect, it may be incompatible with other modules, causing the entire software to fail to run. This is the purpose of testing with caution.

* 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

The later a bug is discovered, the more expensive it is to fix. It may only take a few minutes to fix it during the development phase, but discovering it after deployment may lead to downtime, data loss, and customer complaints. At the same time, rough code will make it difficult for future developers to understand, test, and modify it. To avoid this from happening, the first thing is attitude, and practitioners must be cautious. The second is to avoid subjectivity and use many test software for all-round testing. Third, strictly follow the agile method to test and accept every step and every module of the software.