

7-2 Project Two: Summary and Reflections Report

Southern New Hampshire University



Barbara McKenney

[CS-320-R4838 Software Test Automation& QA](https://learn.snhu.edu/d2l/home/1535936)

Professor Kalysa Wilson

April 21, 2024

**Summary and Reflections Report**Your supervisor has asked that you submit a follow-up summary and reflections report to explain how you analyzed various approaches to software testing based on requirements and applied appropriate testing strategies to meet requirements while developing the mobile application for the customer. This report should be based on your experience completing Project One. You must complete the following:

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The Unit testing approach I took for each feature is as follows:

For the Contact Service feature, I wrote unit tests for each functionality of the ContactServiceClass, including adding, deleting, and updating contacts, and tested each requirement specified for the Contact class and Contact Service. My approach to the Contact Service feature aligned with the software requirements in that each unit test corresponded to a specific requirement outlined for the Contact class and the Contact Service. Furthermore, the test cases covered scenarios where contacts are added, deleted, and updated, ensuring compliance with the state requirements. Example test cases include the test for adding a contact with a unique ID, a test for deleting a contact by contactId, and a test for updating contact fields (firstName, lastName, PhoneNumber, Address) by contactId.

My approach for the Task Service feature was to develop unit tests to validate the functionality of the TaskService class focusing on adding, deleting, and updating tasks and verifying each requirement specified for the Task object and Task Service. My approach to the Task Service feature aligned with the software requirements in that each unit test is designed to validate a particular requirement set for the Task object and Task Service. Additionally, each test case ensured that tasks were added with unique IDs, deleted by taskId, and updated fields (name, description) by taskId. The three test cases in TaskServiceTest (a test for adding a task with a unique ID, a test for deleting a task by taskId, and a test for updating the task field) are examples of how my approach aligned with the software requirements.

My approach for the Appointment Service feature was to create unit tests that ensured the proper functioning of the AppointmentService class by focusing on adding, deleting, and validating appointments and by validating each requirement outline for the Appointment object and Appointment Service. My approach to the Appointment Service feature aligned with the software requirements in that the unit tests were structured to verify the compliance of the AppointmentService with the specified requirements for the Appointment object and services. Additionally, the test cases covered scenarios where appointments are added with unique IDS, deleted by appointmentId, and appointments validated for date and description. An example can be seen in the test that adds appointments with a unique appointmentId, the test that deletes appointments by appointmentId, and the test that validates appointments for date (not in the past) and description length.

* + 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?

The quality of my Junit tests can be defended based on the following factors:

* Test Coverage: Each of the requirements outlined in the software specifications document is covered by one or more-unit tests. By measuring test coverage, I was able to identify gaps in testing.
* Test Assertions: I used appropriate assertions to validate the expected behavior of each functionality being tested.
* Test Maintainability: I wrote modular and easily understandable test cases that could be updated or extended as requirements change.
* Test Readability: I ensured that each test case was well-documented with comments explaining the purpose of each test and the expected outcome.

Test Effectiveness based on Coverage Percentage:

* The high percentage of code coverage my unit tests achieved indicates that more parts of the code were tested, increasing confidence in the reliability of the software. In the image below we can see that the tests package containing all the JUnit tests achieved a test coverage percentage of 95.0 %. That is all Junit tests achieved 100.0 % except AppointmentTest which has a test coverage of 75.0 %.

A screenshot of a computer

Description automatically generated

* I analyzed coverage reports to identify areas of the code that are not adequately covered by tests and prioritized writing additional test cases for those areas.
  + 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?
  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.

I ensured the technical soundness of Junit tests by following best practices, that is, using proper naming conventions, proper setup and teardown, and accurate assertions. For example, in the TaskServiceTest class, the setUp() method is used to initialize a new TaskService instance before each method, ensuring a clean state for each test.

A white background with black text

Description automatically generated

Additionally, assertions were used to verify the expected behavior of the code under test. For instance, in the testAddTask() method, an assertion is used to compare the added task with the task retrieved from the TaskService to ensure they are equal.

A black and blue text

Description automatically generated

In the ContactServiceTest class, an example of technical soundness can be seen in the test method setup. For example, the testAddContact() method (seen below) demonstrates soundness by properly setting up the test environment before executing the test logic. It creates a new instance of ContactService, adds a contact, and then asserts that the added contact can be retrieved successfully.

A computer code with text

Description automatically generated with medium confidence

In AppointmentServiceTest class technical soundness was maintained the existing appointment deletion test case (shown below)

A computer program with text

Description automatically generated with medium confidence

The above test method demonstrates technical soundness by creating an appointment, adding it to the AppointmentService, deleting it, and then verifying that the appointment has been successfully removed by checking that the retrieved appointment is null.

Overall, my code exhibits technical soundness by adhering to best practices, such as proper setup, accurate assertions, and thorough testing of ContactService, TaskService, and AppointmentService functionalities. and ensuring that each test method followed a clear structure and purpose.

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

I ensured the coding efficiency of my JUnit test by writing concise and focused test cases that covered specific functionalities without unnecessary duplication or complexity. For example, in the TaskServiceTest class, each test method focuses on a specific functionality of the TaskService class, such as adding a task, updating task name or description, and deleting a task. This approach keeps the tests modular and maintainable:

A screenshot of a computer program

Description automatically generated

(**Note: To save space, I removed the test logic for the three methods shown above.**)

Another example of code efficiency can be seen in the focused and concise test methods found in the ContactServiceTest class:

A screenshot of a computer

Description automatically generated

This testDeleteContact() method verifies the deleteContact() method’s behavior efficiently by adding a contact, deleting it, and asserting that it cannot be retrieved, all within a single test method.

Similarly, in the AppointmentSeviceTest class, I demonstrated code efficiency in the testAddAppointment\_ValidAppointment\_AppointmentAdded() method:

A screenshot of a computer program

Description automatically generated

This above method succinctly tests the addAppointment() method by adding an appointment and asserting that it can be retrieved successfully, without unnecessary duplication or complexity.

Additionally, technical soundness was maintained through proper setup and assertions and ensured code efficiency through focused and concise test methods, the JUnit tests for the Contact Service, Task Service, and Appointment Service contributed to the overall quality and reliability of the software. Furthermore, by using appropriate setup and teardown methods I prevented test pollution and ensured that each test case was executed independently which contributed to the efficiency of the test suite. Code efficiency of the Junit tests was further maintained by writing focused and streamlined test cases and best practices for test setup and teardown.

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

The two testing techniques employed for this project were unit testing and boundary value analysis. Unit testing is used to test individual units or components of software. In this project, J unit tests were employed extensively for the contact service, Task service, and Appointment Service to verify their functionalities and ensure that each behaves as expected. Boundary value analysis is a technique that was implicitly used in the unit test to verify the behavior of the software at the boundaries of valid and invalid input ranges. For example, testing the behavior of the Contact and Task objects with minimum and maximum valid values for their fields.

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

For this project, unused testing techniques included Integration testing and regression testing. Integration testing was not explicitly used in this project. Integration testing involves testing the interactions between different units or components of the software to ensure they work together as expected. While some aspects of regression testing were employed during the development process, it was not specifically addressed in the provided project context. Regression testing involves retesting the software after changes to ensure that existing functionalities are not adversely affected.

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

I've mentioned two used techniques, Unit testing and Boundary Value analysis, and two unused techniques Integration testing and regression testing. Below I will discuss the practical uses and implications of each technique.

Unit Testing: Unit testing is crucial for identifying defects early in the development process, ensuring that individual units of code behave as expected, and providing a safety net for code changes. It helps in maintaining code quality, reducing the likelihood of introducing bugs in supporting code refactoring baby efforts.

Boundary Value Analysis technique that helps in identifying defects of the boundaries of valid invalid input ranges where errors are more likely to occur. By testing boundary conditions, developers can ensure that their software behaves correctly in education and handles exceptional scenarios effectively.

Integration testing is essential for verifying the different units or components of the software. Work together seamlessly. It helps in identifying interface defects, data flow issues, and compatibility problems between different modules, ensuring the overall correctness and robustness of the software system. Finally, regression testing is critical for ensuring changes to the software do not introduce new defects or regression in existing functionalities. By retesting the software after modifications, developers can catch unintended side effects and maintain the reliability and Instability of the application over time.

* 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.

In working on this project, I adopted a cautious mindset, recognizing the complexity and interrelationships of the code being tested. I understood the importance of thoroughly assessing the functionalities and behaviors of the software to ensure its correctness and reliability. It was important to appreciate the complexity of the code to understand its various interactions and potential points of failure. For example, when testing the appointment service, I carefully considered the validation logic for appointment dates to ensure the past dates were rejected correctly, avoiding potential issues with scheduling future appointments.

* + 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.

To limit bias in my review of the code, I approached testing with an objective and methodical mindset. I focused on identifying potential defects in verifying the adherence to specified requirements without preconceived notions or assumptions, for instance when testing the task service, I checked for updates to task fields based on the defined requirements, regardless of personal preference or opinions.

* + 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.

Being disciplined and committed to quality is essential for software engineering professionals to deliver reliable and maintainable software products. Cutting corners and writing or testing code can lead to the accumulation of technical debt, resulting in increased complexity, reduced productivity, and higher maintenance costs over time. I planned to prioritize clean code practices, including writing clear and well-constructed code, conducting thorough testing and regularly refactoring code as needed. By addressing issues promptly and maintaining a focus on quality throughout the development process, I aim to minimize technical debt and deliver high-quality software solutions to clients.

Let’s delve into specific examples from the three Junit test classes to illustrate the importance of being disciplined and my commitment to quality and avoiding technical debt.

In the test case shown below, I ensured that a contact could be successfully added to the ContactService. By rigorously testing the functionality of adding a contact with valid data, I contributed to the reliability and correctness of the ContactService.

A computer screen shot of a computer code

Description automatically generated

In the below test case, testUpdateTaskName() verifies the functionality of updating the name of a task in the TaskService. By ensuring that task modifications are handled correctly, we contribute to the overall quality and reliability of the TaskService.

A computer screen shot of a task

Description automatically generated

The test case shown below, addAppointment\_ValidAegurments\_ObjectCreated() validates the creation and retrieval of appointments in the AppointmentService. By meticulously testing the addition of appointments with valid arguments, I ensured the correctness and integrity of the appointment management functionality.

A screenshot of a computer program

Description automatically generated

In conclusion, being disciplined in the commitment to quality as a future software engineering professional is crucial to avoid technical debit and ensure the reliability and maintainability of software systems. Cutting corners in writing or testing code can lead to various negative consequences, including increased technical debt reduced software quality, and diminished customer satisfaction. By following best practices, conducting throughout testing, and adhering to coding standards, practitioners can minimize technical debt and deliver high-quality software solutions that meet the needs of stakeholders effectively.

**References**

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.