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

My approach for unit testing each of the three features involved using the software requirements to develop the unit tests. The requirements provided specific criteria for the classes, class methods and class variables, so I used them when creating the code and unit tests. An example of the requirements was for the task class for the variables which included that the task id cannot be longer than 10 characters, cannot be null and cannot not be updated. The task service class requirements for the methods were to be able to add tasks with a unique ID, delete tasks per task ID, and update task name and description fields per task ID. When testing the classes I first created a new object of each class with a set of arguments and then used assertions to check that the getter methods of the object return the expected values. This initial test was designed to verify whether the constructor initializes the object correctly. I then proceeded to test for various invalid cases, such as passing null values or values that exceed the maximum length for a parameter, by testing whether the constructor throws the expected IllegalArgumentException. Then I designed unit tests to verify the correct behavior of the methods in each class. These tests included adding a new contact, deleting a task, and adding a new appointment. To confirm the expected results, I used assertions to compare them against the actual results obtained from running the methods. The JUnit tests were effective because they achieved a high coverage percentage when run. Test cases were built based on the requirements, checking the properties of variables and class methods. Object instances of the classes were created, and their methods were tested to ensure proper functionality. Overall, these tests confirm that the code behaved correctly in different scenarios.

1. Testing Techniques

The software testing technique I employed in the project was unit testing. I used the requirements for the classes and created tests for the individual units of the software to ensure that each one works as intended. The first test I used for the classes was creating a new object of the class with a set of arguments, and then asserts that the getter methods of the object return the expected values. Then I would test whether the constructor throws the expected IllegalArgumentException when an invalid argument is passed. Finally, for the classes of the milestones I designed unit tests to verify the correct behavior of the methods in the classes. In the milestones, I did not use performance testing, a software testing technique used to assess software system performance under varying workloads. For instance, in the Contact milestone, performance testing could have been carried out by adding, deleting, or updating many contacts simultaneously to determine factors such as response time and error rates. This would have helped to ensure that the system could handle the expected workload and still function correctly. Unit testing is a valuable software testing technique that enables developers to test their code in isolation and ensure that individual components function as expected. This technique is beneficial in various software development projects and situations. For example, it aids agile development as code can be continuously tested early and often, ensuring that issues are caught early in the development cycle. Additionally, unit testing is useful in continuous integration because the code can be automatically tested to ensure that it functions properly and meets standards.

1. Mindset

The extent that I employed caution was not making assumptions about the software's behavior and thoroughly testing each class to ensure it was functioning as expected. I made sure to correct the issues I found in the code based on the J-Unit tests when receiving error messages. When running the JUnit test for my contact class I ran a test to check that the id couldn’t be longer than 10 characters and it failed. I had made a mistake when creating the verifyID method and put that the id length needed to be smaller than 20 instead of 10 generating the error. Also, I interpreted the coverage percentage and updated the tests to more thoroughly test the units. It was important to appreciate the complexity and interrelationships of the code because it helps in finding potential areas of the software that could be affected by other parts of the code. This helped in developing the proper unit tests to fit the requirements given. I limited bias in the review of my code by creating the tests cases based on the requirements given for the application. I used the requirements to structure the unit tests, and check that the functionality matched the requirements given. I also reviewed the coverage percentage to identify areas of the code that were not covered by the tests, allowing me to improve the coverage and reduce the risk of bias. There can be bias when testing your own code because you may unintentionally overlook errors in the code. Also, developers may be hesitant to report issues that they have found themselves, as they may worry about the implications from the mistakes. It’s important to be disciplined in your commitment to quality as a software engineering professional because it benefits your reputation and helps create user satisfaction. When designing quality software, putting the user's needs first in development ensures that users have a pleasant experience with the product, which can lead to better user satisfaction and loyalty. Also developing quality products creates a reputation for quality software which leads to more business and attracts new customers. I plan to avoid technical debt by following the best coding practices such as writing clean code, using automated testing, and constantly reviewing my code.