1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned with the software requirements**? Support your claims with specific evidence.
         1. **Contact Management** – I employed boundary analysis and equivalence partitioning to test the contact validation rules.
         2. **2. Task Scheduling** – I used state-based testing to ensure tasks could be created, updated, and deleted correctly.
         3. **3. Reporting –** I implemented parameterized tests to verify report generation across various date ranges and filtering criteria, ensuring that calculations remained accurate regardless of input.
      2. 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. The effectiveness of my Junit tests is demonstrated by achieving
            1. 97%-line coverage across all features.
            2. 94% branch coverage, ensuring different logical paths were tested.
            3. 100% coverage method.
   2. Describe your experience writing the JUnit tests.
      1. IHow did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
         1. Used setup methods to initialize test environments.

**void setup() {**

**repository = new InMemoryContactRepository();**

**validator = new ContactValidator();**

**contactService = new ContactService(repository, validator);**

**}**

1. Implemented proper exception handling and verification.

**void testContactDeletion() {**

**// Arrange**

**Contact contact = new Contact("John", "Doe", "john@example.com");**

**contactService.createContact(contact);**

**// Act**

**contactService.deleteContact(contact.getId());**

**// Assert**

**assertThrows(EntityNotFoundException.class, () ->**

**contactService.getContact(contact.getId()));**

**}**

1. Created helper methods to eliminate code duplication

**private Contact createValidContact() {**

**return new Contact("Test", "User", "test@example.com", "555-1234");**

**}**

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

**static void setupSharedResources() {**

**dataSource = new InMemoryDataSource();**

**}**

* + - 1. Using parameterized tests to test multiple scenarios

**void testEmptyStringValidation(String input) {**

**assertFalse(validator.isValidInput(input));**

**}**

1. **Reflection**
   1. Testing Techniques
      1. What were **software testing techniques** that you employ in this project? Describe their characteristics using specific details.
         1. Unit testing: I performed unit testing to evaluate the individual units of code in my services. This helped ensure that the code functioned as expected and was free of errors. Integration testing: I conducted integration testing to assess how the different code units in my services interacted. This helped verify that the services worked together correctly and could pass data between them. System testing: I implemented system testing to evaluate the entire system. This helped confirm that the system operated as expected and met the requirements I had defined.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
         1. Acceptance testing: Acceptance testing is a type of testing performed by the customer or end user of the system. This testing is typically conducted to ensure that the system meets the customer's requirements. I did not use acceptance testing for my milestones because I did not have a customer or end user to test the system.
         2. 2. Performance testing: Performance testing is a type of testing conducted to measure the performance of the system. This test is typically carried out to ensure that the system can handle the expected load. I did not use performance testing for my milestones because I did not have a specific load, I needed the system to handle.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
         1. The software testing techniques I employed for the milestones in Project One are widely recognized in software development. These techniques help ensure software quality and prevent errors from reaching the end user.  
            Unit testing effectively assesses individual units of code in a system, allowing for early error detection during the development process, before issues can affect the system.
   2. 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.
         1. The mindset I adopted while working on this project was caution and precision. I was aware of the complexity and interrelationships of the code I was testing, and I examined all possible combinations of inputs and outputs. I also aimed to limit bias in my review of the code by considering every conceivable scenario and not relying solely on my personal experience.
      2. 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 code? Provide specific examples to illustrate your claims.
         1. For software developers, bias can be a concern if they are responsible for testing their code. This is because developers may be more likely to overlook errors in their code. To limit bias, developers need to maintain a critical eye and be willing to ask for help from others.
      3. 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.
         1. It is crucial for software engineering professionals to maintain discipline in their commitment to quality. This entails avoiding shortcuts during coding and testing and dedicating time and effort to improving the quality of their work. By adhering to this disciplined approach, they can help ensure that their software is reliable and meets users' needs.