1. Task: Understanding the Phases of SDLC

- **Objective**: Understand and identify the phases involved in the Software Development Life Cycle (SDLC).
- **Steps**:
    1. Study the six phases of SDLC: Requirement Gathering, System Design, Implementation, Testing, Deployment, and Maintenance.
    2. Choose a software project you are familiar with and map it to the SDLC phases.
    3. Write a brief description of each phase as applied to the chosen project.
- **Question**:
  How does the Testing phase in SDLC contribute to the overall quality of the software, and what types of testing should be performed?

---

2. Task: Applying Agile Methodology to a Project

- **Objective**: Learn how to apply Agile methodology in real-world software development.
- **Steps**:
    1. Research Agile principles and the Scrum framework.
    2. Break down a sample software project (e.g., a task management tool) into smaller features or user stories.
    3. Organize the features into sprints and create a backlog.
    4. Conduct a sprint planning meeting and outline the tasks for the first sprint.
- **Question**:
  How do Agile practices improve collaboration between developers and stakeholders during software development?

---

3. Task: Implementing Test-Driven Development (TDD)

- **Objective**: Learn how to write tests before writing code using Test-Driven Development (TDD).
- **Steps**:
    1. Choose a simple feature to implement (e.g., a function that calculates the total price of items in a shopping cart).
    2. Write a test case that checks the correct output for various inputs.
    3. Write the code to pass the test case.
    4. Refactor the code if necessary while ensuring that the test passes.
- **Question**:
  What are the advantages of using TDD in ensuring code reliability and maintaining long-term software quality?

---

4. Task: Performing Unit Testing on a Function

- **Objective**: Understand and apply unit testing on a small, isolated piece of code.
- **Steps**:
    1. Choose a function to test (e.g., a function that validates user input).
    2. Write test cases that cover different scenarios (valid input, invalid input, edge cases).
    3. Run the tests and fix any failing tests by modifying the function.
- **Question**:
  Why is it important to write unit tests for individual functions, and what is the impact of failing to test them on the software's reliability?

---

5. Task: Implementing Continuous Integration (CI) in Agile Projects

- **Objective**: Learn how to integrate Continuous Integration (CI) into an Agile workflow to improve code quality.
- **Steps**:
    1. Set up a CI tool (e.g., Jenkins, GitLab CI).
    2. Configure the tool to automatically run tests each time code is pushed to a version control system (e.g., Git).
    3. Push code changes and observe how the CI tool runs automated tests.
- **Question**:
  How does Continuous Integration help in maintaining a stable and high-quality codebase throughout the development lifecycle?

---

6. Task: Performing Integration Testing

- **Objective**: Understand how integration testing ensures the cooperation of different software components.
- **Steps**:
    1. Identify two or more components in a system (e.g., user authentication and payment system).
    2. Write integration test cases to ensure the components work together correctly.
    3. Execute the integration tests and analyze the results.
- **Question**:
  How do integration tests differ from unit tests, and why is integration testing essential for software systems with multiple interacting modules?

---

7. Task: Refactoring Code to Adhere to the Single Responsibility Principle (SRP)

- **Objective**: Refactor a class to ensure it follows the Single Responsibility Principle (SRP).
- **Steps**:
    1. Review an existing class that performs multiple tasks (e.g., a class that handles both user authentication and logging).
    2. Break the class into two smaller classes, each with a single responsibility (e.g., one class for user authentication, one for logging).
    3. Test the refactored classes to ensure they function correctly and meet the desired outcomes.
- **Question**:
  How does adhering to SRP make the code more maintainable, and what are the potential challenges of refactoring a class that violates SRP?

---

8. Task: Designing a System Using the SDLC Phases

- **Objective**: Practice designing a system by following the SDLC process.
- **Steps**:
    1. Choose a small project, like a to-do list app, and outline its requirements.
    2. Design the system architecture, including the database structure and API endpoints.
    3. Create a timeline for development, testing, and deployment, following the SDLC phases.
- **Question**:
  What are the most common challenges teams face during the Design phase of SDLC, and how can they be mitigated?

---

9. Task: Conducting User Acceptance Testing (UAT)

- **Objective**: Perform User Acceptance Testing (UAT) to validate that the software meets the user's needs.
- **Steps**:
    1. Select a project (e.g., an inventory management system).
    2. Work with the end-users to define acceptance criteria for each feature.
    3. Run through the features and ensure they meet the criteria.
    4. Document any issues and work with developers to address them.
- **Question**:
  How does User Acceptance Testing differ from other types of testing, and why is it crucial to get feedback from end-users before releasing a product?

---

10. Task: Applying Agile Retrospective to Improve Team Performance

- **Objective**: Learn how to conduct an Agile retrospective to identify ways to improve team collaboration and performance.
- **Steps**:
    1. After completing a sprint, gather the development team for a retrospective.
    2. Discuss what went well during the sprint, what could be improved, and any obstacles faced.
    3. Identify actionable items for the next sprint to improve team performance and delivery.
- **Question**:
  How do retrospectives help Agile teams continuously improve, and what are some common challenges in conducting effective retrospectives?