Assignment: Mini Software Engineering Project – Smart Event Reminder System (SERS)

Objectives

Apply the complete software engineering process to a small system, with emphasis on: - Agile methods and user stories. - System modeling using sequence and state diagrams. - Architecture selection and justification. - Implementation quality and testing. - Short iteration and reflection.

System Description

Design a simple system called **Smart Event Reminder System (SERS)** that allows users to manage events and receive reminders.

Main Functions

- Create, edit, and delete events.
- Add event details (title, description, date, time, optional location).
- Send reminders before events occur.
- (Optional) Invite other users or share events.

Scope

- Concept only — no full implementation or database needed.
- Focus on documentation, process, and code quality.
- Mock data or pseudo-code is acceptable.

Stakeholders

Stakeholder	Role
User	Creates and manages events.
Admin	Manages user accounts and system data (optional).
Notification Service	Triggers reminders.
Project Team	Designers/developers of the system.

Stakeholder	Role
Instructor	Evaluates software engineering process and documentation.

🗒 1. Requirements Engineering

Goal: Identify what the system does and who uses it.

Tasks

1. List and describe all **stakeholders**.
2. Define at least **6 Functional Requirements (FRs)** and **3 Non-Functional Requirements (NFRs)**.
3. Write **at least 5 User Stories** using this format:
   As a [type of user], I want [goal] so that [benefit].
4. Add **acceptance criteria** for each story (how to verify it works).
5. Prioritize using **MoSCoW** (Must, Should, Could, Won't) table.
6. (Optional) Create a small **requirement traceability table** linking FRs → User Stories → Diagrams.

Expected length: 1–2 pages.

⚙ 2. Process Model and Planning

Goal: Show how the project will be developed and managed.

Tasks

1. Choose one model: **Scrum**, **Waterfall**, or **Incremental**.
2. **Justify** why it fits your team and project.

If Scrum

- Create a **Product Backlog** and **Sprint Backlog**.
- Estimate effort with **Planning Poker** (0–21 scale).
- Create a **Burndown Chart** (simulated is fine).

If Waterfall

- Draw a timeline showing each phase: Requirements → Design → Implementation → Testing → Maintenance.
- Describe deliverables from each phase.

If Incremental

- Define **3 increments** (e.g., event creation, reminder scheduling, sharing).

- List what each increment adds and how it will be tested.
- Draw a simple **incremental roadmap** showing how functionality grows.

Expected length: 1–2 pages.

---

3. System Modeling

Goal: Model structure and behavior of the system.

Tasks

1. Draw **three UML diagrams:**
    - **Sequence Diagram** – e.g., user creates a new event.
    - **State Diagram** – event lifecycle (Created → Scheduled → Notified → Completed/Cancelled).
    - **Class Diagram** – main entities and methods.
2. Keep diagrams clear, consistent with requirements, and labeled properly.

Expected length: 2–3 diagrams with short captions.

---

4. Architecture Design

Goal: Show the high-level structure of your system.

Tasks

1. Select one **architectural pattern** (Layered, MVC, Client-Server, etc.).
2. Draw a **Component Diagram** or high-level **Architecture Diagram**.
3. Explain briefly:
    - Each layer/component's role.
    - How data flows between them.
    - Why this architecture fits your project.
4. (Optional) Suggest a simple **technology stack** (e.g., React + Flask + SQLite).

Expected length: 1 page.

---

5. Implementation and Code Quality

Goal: Show that your design can be turned into clean, maintainable code.

Tasks

1. Choose **one main feature** (e.g., create event or send reminder).
2. Write a **short code sample (~100–150 lines)** to demonstrate it.
3. Apply good coding practices:
   - Clear naming, consistent indentation, comments.
4. Conduct a **code inspection**:
   - Peer-review each other's code.
   - Identify 3–5 small issues (naming, readability, missing checks).
5. Find **two code smells** (e.g., long method, duplicate logic) and describe how you would **refactor** them.
6. Submit:
   - Code listing.
   - Short inspection summary.
   - Refactoring explanation (before/after or description).

Expected length: 1–2 pages.

🧪 6. Software Testing and Quality Assurance

Goal: Design and document tests for key system functions.

Tasks

1. Select **2–3 key functions or user stories** to test.
2. Write a short **test plan** describing:
   - Objective.
   - Test level (unit, integration, system).
   - Test method (black-box or white-box).
3. Create **at least three (3) test cases** using the table format below:

Test ID	Description	Input	Expected Output	Test Type
T1	Create event with valid data	Title="Meeting", Date="2025-10-15"	Event saved successfully	Black-box
T2	Add event with missing title	Empty title	Error message displayed	Black-box
T3	Reminder delivery	Event=10:00, Now=9:5	Notification triggered	System

Test ID	Description	Input	Expected Output	Test Type
		9		

4. (Optional) Add short test evidence or describe how automation could be done.

Expected length: 1 page.

---

7. Reflection and Retrospective

Goal: Reflect on teamwork, process, and lessons learned from doing the project.

Tasks

1. **Hold a short team meeting** at the end of the project to discuss what worked and what didn't. Everyone should contribute their views.
2. Use the **Start / Stop / Continue** framework to structure your discussion:
    - **Start:** What should your team start doing next time? (e.g., plan earlier, divide tasks better, communicate more often)
    - **Stop:** What should your team stop doing? (e.g., last-minute changes, unclear task ownership, skipping reviews)
    - **Continue:** What practices or habits worked well? (e.g., pair-programming, frequent updates, shared drive organization)
3. Summarize your reflection clearly. You can use either a short paragraph or a simple table like this:

Category	Team Reflection
Start	Begin weekly short meetings to check progress.
Stop	Stop committing code without peer review.
Continue	Keep using shared diagrams for consistency.

4. Include one short paragraph explaining **what your team learned overall** about teamwork and software engineering from this project. Mention specific skills or insights (e.g., estimating effort, managing scope, improving diagram consistency, debugging strategies).
5. (Optional) Each member may write 2–3 sentences of personal reflection about their role and contribution.

Expected length: about 1 page total. **Focus:** clarity, honesty, and connection to process improvement.

Final Submission

- Submit **one PDF report (8–10 pages)** including all sections above.

- No working prototype required.
- Team size: 3–4 students.
- Duration: ~2 weeks.

Report Outline

1. System Description
2. Requirements Engineering
3. Process Model & Planning
4. System Modeling
5. Architecture Design
6. Implementation & Code Quality
7. Software Testing & QA
8. Reflection

Evaluation (Total 20 points)

Category	Criteria	Points
Requirements & User Stories	Completeness and clarity	3
Process Model & Planning	Realistic and justified	2
System Modeling	Correct UML diagrams	4
Architecture Design	Logical pattern and explanation	3
Implementation & Code Quality	Structure, inspection, refactoring	3
Testing & QA	Test coverage and clarity	3
Reflection	Team insight and reflection	2
