National University

of Computer & Emerging Sciences, Islamabad

Department of Software Engineering

(Software Reengineering)

Project

Semester: FALL 2025
Course Instructors:
Ms. Nigar Azhar Butt, Ms. Anum Kaleem, Ms. Fatima Gilani
Sections: All

Instructions

- Read the instructions very carefully.
- Late assignments will not be accepted
- Plagiarism may result in zero marks in the whole assignments category (all assignments) regardless of the percentage plagiarized.
- There will be no credit if the given requirements are changed.
- **Group Size:** 2–3 students

Submission Guidelines

- Submit the report, and all relevant artefacts.
- **Submit improved code base**

1. Project Overview

Students are provided with a legacy **desktop-based** Point-of-Sale (POS) system originally developed in Java. The current system stores its data in plain **.txt files**, which is not scalable or maintainable.

The project follows the Software Reengineering Process Model, progressing through the stages of Inventory Analysis, Document Restructuring, Reverse Engineering, Code Restructuring, Data Restructuring, and Forward Engineering.

The goal is to transform the legacy desktop POS system into a **modern, web-based system** with improved architecture, documentation, data management, and maintainability, demonstrating the ability to apply each phase of the reengineering process systematically.

---

2. Project Aim

To perform complete software reengineering of the given legacy POS system by applying all stages of the model:

1. Inventory Analysis – identify, classify, and assess existing software assets.
2. Document Restructuring – reorganize or recreate technical documentation.
3. Reverse Engineering – recover design, data structures, and logic from the existing codebase.
4. Code Restructuring – improve maintainability and readability of the legacy code.
5. Data Restructuring – redesign and migrate data from plain-text storage to a **proper database** with a well-designed schema.
6. Forward Engineering – develop and deliver an improved version of the system as a **web-based application** using modern technologies and architecture.

---

3. Technology and Language Choice

The reengineered system must be implemented as a **web-based application**.

Students may use any programming language, web framework, and database based on their expertise and project goals (for example: Python + Django, C# + ASP.NET, Java + Spring Boot, Node.js + Express, etc.).

Since the original system uses .txt files for data persistence, students must redesign the data model and migrate all data into a **database of their choice** (relational or NoSQL). The decision must be justified.

All technology decisions must be explained in the report, including:

- Why the chosen programming language is suitable
- Why the selected framework supports improved functional or non-functional requirements

- Why the chosen database is appropriate
- Necessary schema improvements and their justification

The focus is on architectural improvement, data quality, maintainability, and clarity of design.

---

4. Phased Activities and Expected Outputs

1. **Inventory Analysis** – Identify all code, libraries, configurations, documentation, and tests. Classify assets (active, obsolete, reusable) and map dependencies.
2. **Document Restructuring** – Rebuild or update technical documentation to represent the true system structure.
3. **Reverse Engineering** – Analyze legacy code to recover design, workflows, and data structures. Identify code and data smells.
4. **Code Restructuring** – Apply refactoring and restructuring to improve modularity, reduce complexity, and improve readability.
5. **Data Restructuring** – Redesign and normalize the data model; migrate from .txt files to a proper database with a well-defined schema.
6. **Forward Engineering** – Implement the reengineered POS system as a **web-based application**, ensuring the architecture is modern, modular, maintainable, and improved.

---

5. Improved Architecture

The reengineered system must present an improved architecture that demonstrates:

- Clear separation of concerns
- Layered structure (Presentation, Business Logic, Data Access)
- Maintainable data flow
- Centralized business logic
- Database integration using a repository pattern or ORM
- Configurable and testable modular design
- Use of appropriate design patterns (MVC, Repository, Singleton, Observer, etc.)

Documentation must include both legacy and reengineered architecture diagrams, a comparison table, and clear justification for all design improvements.

---

6. Documentation Requirements

A. Legacy System Documentation (Reverse Engineered)

- System overview and module inventory
- Extracted architecture and class diagrams
- Identified code and data smells
- List of current limitations

B. Reengineered System Documentation (Forward Engineered)

- Updated architecture and design diagrams
- Refactored module and data structures
- Database schema, migration plan, and rationale
- Technology stack selection and justification
- Mapping from legacy components to the new system
- Evidence of improved architecture and maintainability

C. Refactoring Documentation

Each team member must identify, perform, and document at least **three major refactorings** with:

- Before and after code
- Explanation
- Quality impact

---

7. Team Composition and Work Distribution

- Teams of 2–3 members
- Each member must contribute to at least one major phase
- Each member must document three refactorings
- Work distribution must be clearly recorded in a signed table in the report

---

8. Deliverables

1. **Technical Report** covering all model phases, refactorings, and work distribution.
2. **Reengineered Web-Based System Codebase**, including database schema, configuration, and deployment instructions.

9. Guidelines

- Preserve original system functionality; no unnecessary new features
- Justify all architectural, framework, and database decisions
- Use version control
- Ensure consistency and clarity across all documentation
- All improvements must relate back to issues identified in the legacy system

10. Rubric:

Category	Marks	Expectations
1. Inventory Analysis & Document Restructuring	**15**	Complete asset inventory; correct classification; accurate dependency mapping; legacy documentation fully reconstructed with diagrams.
2. Reverse Engineering & Smell Detection	**15**	Strong code understanding; accurate extracted architecture; identifies multiple code/data smells with evidence.
3. Code Restructuring	**10**	Comprehensive refactoring; improved modularity/clarity;
4. Data Restructuring	**10**	normalized schema; well-justified data migration.
5. Forward Engineering (Improved Architecture)	**15**	Fully implemented improved architecture with clear layers; justified tech stack; modular and maintainable system.
6. Reengineering Plan & Migration	**10**	Clear, realistic plan covering all phases (inventory → reverse engineering → restructuring → forward engineering); includes timeline, risk considerations, and accurate migration strategy from old to new architecture/data.
7. Refactoring Documentation (Individual)	**10**	Each member documents 3+ major refactorings with before/after code, rationale, and impact.
8. Risk Analysis & Testing	**10**	Key risks identified with mitigation; strong testing evidence (unit/integration/database tests).
9. Dual Documentation (Legacy ↔ Reengineered)	**10**	Complete comparison with diagrams, mapping tables, and justification of changes.
10. Work Distribution & Team Contribution	**5**	Clear contribution table; tasks and refactorings documented; all members sign.
Total	110	

Best of Luck ☺