Wollo University

Kombolcha Institute of Technology

College of Informatics

DEPARTMENT OF SOFTWARE ENGINEERING

Software Engineering Tools and Practices (SEng5331) Project

1. Overview & Objectives

This project is a group-based assignment designed to apply the core concepts from the **Software Engineering Tools and Practice** course.

This project is a group-based assignment designed to apply the core concepts from the Software Engineering Tools and Practice course. You will work in teams to define, design, and begin implementing a software application of your choice. Each group is free to choose any project idea they want (as long as it meets the course requirements and can be completed within the available time frame).

The focus is not only on the final product, but also on the process, documentation, collaboration practices, and tools used throughout development. Your collaborative work, tracked and managed through GitHub, will be a major component of your grade.

Core Objectives:

- **Analysis:** Identify a system's requirements, actors, and functionalities.
- **Design:** Model a system using UML diagrams (Use Case, Activity, Sequence, and Class diagrams).
- **Tooling:** Use Enterprise Architect.
- **Collaboration:** Manage your project source code using Git and GitHub, following a collaborative workflow.

2. Team Requirements

- All work will be done in **groups of 5 members**.

3. Project Phases & Tasks

This project is divided into distinct phases. Each phase has specific tasks and deliverables.

Phase 1: Formation & Setup

Goal: Form teams, choose a project, and set up your collaborative environment.

1. **Form Groups:** Organize yourselves into groups of 5.
2. **Select Project:** Brainstorm and decide on a software project idea.
   - ❿ **Deliverable:** Submit your group list and a 1-paragraph project proposal.
3. **Setup GitHub Repository:**
   - ❿ One member (the "repo owner") must create a **new repository** on GitHub.
   - ❿ Name it something clear.
   - ❿ Add all other 4 group members as "Collaborators".
4. **Initial Commit:**
   - ❿ The repo owner will create a README.md file in the repository. This file must contain:
     - ❿ Your Project Title
     - ❿ A brief (1-2 sentence) description of the project.
     - ❿ The full names and student IDs of all 5 group members.
   - ❿ The repo owner will make the first commit with this file.
5. **Verify Collaboration:**
   - ❿ **Every other member** must clone the repository, add a small piece of information (like their GitHub username) next to their name in the README.md, commit their change, and push it to the repository. This ensures everyone has access and is set up.

Phase 2: Requirements Analysis & UML Modeling

Goal: Define *what* your system will do and model its behavior.

1. **Identify Actors:** List all the "actors" (types of users or external systems) that will interact with your system.
2. **Define Functional Requirements:** Write a list of user stories or use cases describing the system's functionalities.
3. **Draw Use Case Diagram:** Create one (1) complete Use Case Diagram for your entire system, showing all actors and their relationships to the primary use cases.
4. **Draw Behavioral Diagrams:**
   - ❿ Select **three (3) to four (4) of your most important use cases**
   - ❿ For *each* of these selected use cases, you must draw:
     - ❿ **Activity Diagram** showing the flow of events.
     - ❿ **Sequence Diagram** showing the object interactions.
5. **Deliverable:**
   - ❿ Create a folder named docs in your repository.
   - ❿ Add all diagrams (as images like .png or .jpg) and your list of actors/requirements (in a Word or Markdown file) to this folder.

- Commit and push these files. This work should be divided among group members.

Phase 3: System Design & Code Engineering

Goal: Design the system's static structure and practice round-trip engineering.

1. **Draw Class Diagram:** Create one (1) detailed **Class Diagram** for your entire system.
   - It must show all major classes.
   - For each class, include key attributes (with data types) and methods (with parameters and return types).
   - You must show relationships: **Association, Aggregation, Composition, and Inheritance** where appropriate.
2. **Forward Engineering (Model to Code):**
   - Using **Enterprise Architect**, import/re-create your Class Diagram.
   - Use the tool's "Code Generation" feature to generate the class files (in Java).
   - **Deliverable:** Create a /src folder in your repository and commit all the auto-generated code files. The commit message must be "Initial code generation from Class Diagram."
3. **Reverse Engineering (Code to Model):**
   - Manually edit 2-3 of the generated classes. Add constructors, getters/setters, or a simple method body.
   - Commit these manual changes with a clear message (e.g., "Added getters/setters to User and Book classes").
   - Now, use the "Reverse Engineering" feature in Enterprise Architect to import your *modified* code.
   - **Deliverable:** Create a short report (Phase_2_Report.md) in your docs folder. This report must include screenshots showing:
     1. Your final Class Diagram in the tool.
     2. A snippet of the auto-generated code.
     3. A snippet of your manual code modifications.
     4. The updated Class Diagram after reverse engineering the changes.

Phase 4: Collaborative Implementation & Version Control

Goal: Work as a team to implement the system, using Git/GitHub for every step.

1. **Task Breakdown:** As a group, look at all the methods in your Class Diagram that need to be implemented (e.g., loginUser(), calculateFine(), etc.).
2. **MANDATORY Git Workflow:** You must follow this process for *all* work in this phase.
   - **a. Create a Branch:** Before starting a task, the assigned member must create a new branch from main. The branch should be named after the name of the student.
   - **b. Commit Your Work:** As you work on your task, make small, frequent commits. Each commit must have a clear message (e.g., "Add validation for user password").
   - **c. Open a Pull Request:** When your task is finished and tested, push your branch to GitHub and open a **Pull Request (PR)**. In the PR description.
   - **d. Code Review:** At least **one other group member** must review the Pull Request.
   - **e. Merging:** Only after the PR is reviewed and approved can it be merged into the main branch.

Phase 5: High-Level Design (HLD) and Low-Level Design (LLD)

HLD: In high-level design you are requested to-

1. Design the overall system architecture diagrams for your project (Provides a **big-picture** view of the system)
2. Module diagram (define modules, subsystems and interfaces).
3. Data flow diagram (how data move through the system).
4. High-level database schema

LLD: In this phase your focus is on the internal structure of each component, including **classes, methods, logic, and database details**

1. Class diagram method definitions
2. Pseudocode or algorithms
3. Detailed database tables
4. Sequence diagrams
5. Error handling and validations

Phase 6: Unit Testing with Junit

You have to apply the following key features of JUnit test

1. Automated test execution
2. Readable and structured test cases
3. Supports assertions for validation
4. Supports Test-Driven Development (TDD)

4. Grading & Evaluation

Your project grade will be based on both group and individual performance.

Group Grade (---%):

- **Completeness & Quality:** Are all required tasks are present and correct?
- **Code Engineering:** Was the forward/reverse engineering process completed and documented correctly?
- **Design Quality:** Was the system well organized and technically sound?
- **Testing:** Does it ensure code quality, simplifies debugging, support maintainability and automates regression testing**?**
- **Project Management:** Does the GitHub repository (PRs, project board) show good organization?

Individual Grade (---%):

- This grade is determined *entirely* by your activity in the GitHub repository.
- **Contribution:** We will use the GitHub "Insights" and "Commits" logs to see:
  - **Quantity of Commits:** Did you contribute code regularly, or just one large commit at the end?
  - **Quality of Commits:** Are your commit messages clear? Is your work spread across logical branches?
  - **Collaboration:** Did you participate in the workflow? (i.e., Did you open Pull Requests?)

A note on collaboration: If one member does all the work and pushes it directly to main, the group will get a group grade, but that one member will get individual grade while the others will not be evaluated. **You MUST follow the branch/PR workflow to prove your contribution.**