Software Architecture

(SCS 2303)

Assignment 3

(Release date: 06 August 2025, Deadline: 06 September 2025)

1. Introduction

The objective of this assignment is to demonstrate the ability to translate a set of given business requirements into a robust, maintainable, and scalable software design by applying appropriate software design principles, design patterns and architectural patterns.

This objective breaks down into several key aspects:

- **Translation of Requirements**: It emphasises the core skill of understanding and interpreting business needs expressed in requirements documents and transforming them into a technical representation suitable for software development. This involves identifying key functionalities, data structures, and user interactions.
- **Application of Principles**: The assignment should explicitly require students to demonstrate their understanding and application of software design principles like SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), DRY (Don't Repeat Yourself), and KISS (Keep It Simple, Stupid). Students should be able to justify their design choices based on these principles.
- **Maintainability**: The design should be easy to understand, modify, and extend in the future. This is achieved through modularity, clear separation of concerns, and the application of design principles that promote code clarity and reduce complexity.
- **Scalability**: The design should be able to handle increasing workloads and data volumes without significant performance degradation. This might involve considering and selecting suitable architectural patterns and using appropriate data storage and retrieval strategies.
- **Robustness**: The design should be resilient to errors, handle unexpected inputs gracefully, and be able to recover from failures. This might involve incorporating error-handling strategies, input validation, and redundancy mechanisms.

2. Duration and Team Composition

The assignment duration is 4 weeks (deadline: 6 September 2025) and should be completed in teams of **8–9 students**.

3. Submission Guidelines

The assignment submission should comply with the following criteria.

1. A document completed with the software design analysis containing the overall architecture, UML diagrams, Object-Oriented Design, and use of Design Patterns.

2. The design pattern implementation source code should be provided with the submission. Students are allowed to select a programming language out of the following options: C++, C#, Java or Python.

3. A screencast video of the functionality of the implemented solution should be recorded and included with the submission. The video should not exceed 10 minutes of recording time and should focus on the interaction of the solution using the user interfaces or test cases to interact with the business logic tier.

4. The assignment documentation, screencast video and source code (compressed file) should be uploaded to the VLE on or before the submission deadline.

4. Assignment Outline and Tasks: University Course Enrolment System – A Modernisation Project

Part A: Architectural Design (30% of grade)

1. **Select Architectural Patterns:** Choose suitable architectural patterns (one or a combination of patterns to complement the overall solution architecture) for the NexusEnroll system. Your options are:
   - Microservices Architecture
   - Service-Oriented Architecture (SOA)
   - 3-Tier Architecture
2. **Justify Your Choice:** Provide a detailed justification for your chosen architectural pattern. Explain why it is the best fit for the university's needs, considering factors like scalability, maintainability, and the ability to integrate with future services (e.g., a new financial aid system).
3. **Draw and Describe the Architecture:** Create a clear diagram of your proposed architecture. Label all the layers, tiers, services, and their communication pathways. For each major component, briefly describe its function and responsibilities.

Part B: Detailed Design & Implementation (70% of grade)

1. **Identify Core Features:** Focus on the core modules related to student, faculty and administrator as described in section 5.
2. **Apply Design Patterns:** Within the context of your chosen architectural pattern, design and implement the logic for this feature using a minimum of **three distinct object-oriented design patterns**.
   - You must identify the patterns you are using.
   - For each pattern, explain where and why you are using it. For example, "I am using the **{PATTERN}** to notify the student's academic advisor and the billing system whenever a student successfully enrols in a course."
3. **Proposed Design Pattern Examples:**
   - **Creational:** Factory Method, Singleton, Abstract Factory, Builder, etc.

- **Structural:** Adapter, Facade, Decorator, Proxy, etc.
- **Behavioural:** Strategy, Observer, Command, State, Template Method, etc.

4. **Core Business Logic Design:**
    - Focus your design efforts on the **core business logic layer**. Exclude presentation and other functional tiers.
    - Identify the primary objects within the business tier and define the relationships between them.
    - Analyse the object-oriented model of the business tier using UML diagrams:
        - Activity Diagram: Model the flow of key business processes.
        - Sequence Diagram: Illustrate interactions between objects in specific scenarios.
        - Collaboration Diagram (Optional Alternative to Sequence Diagram): Show object interactions with a focus on object roles.
        - State Diagram: Model the lifecycle of key objects.
        - Class Diagram: Depict the structure and relationships of classes.
5. **Software Design Principles:**
    - Ensure your final design adheres to widely accepted software design principles (Encapsulation, Programming to an Interface, Composition over Inheritance, SOLID principles, etc).
    - Provide at least one concrete example in your design demonstrating adherence to each principle.
6. **Implement the Solution:** Write a small, runnable proof-of-concept application (using a language like C++, Java, C#, or Python) that demonstrates your design. The code should:
    - Implement the modules – student, faculty and administrator.
    - Represent the core components you defined in your architecture.
    - Show how the design patterns are integrated into the three modules' logic.

   - Include a simple main function or test classes that simulate student, faculty and administration functions/user stories – implementation of UI to interface the core business tier logic is optional.
   - Provide clear comments explaining the code and the role of each design pattern.

7. **Documentation:** Submit a report that includes:
   - Your architectural diagrams and justifications.
   - Class diagrams for your features, highlighting the design patterns used.
   - Descriptions of each design pattern's role and implementation.
   - Your final, well-commented source code.

Assessment Criteria:

- Architectural Choice & Justification (15%)
- Architectural Diagram & Description (15%)
- Design Pattern Application (40%)
- Implementation & Code Quality (20%)
- Documentation (10%)

5. Requirement Summary: University Course Enrolment System - A Modernisation Project

1. Business Scenario:

The "Nexus University" is a large, public university with a rapidly growing student body. Their existing course enrolment system, "LegacyEnroll," is a monolithic application that has been in use for over 20 years. It's difficult to maintain, slow, and lacks modern features like real-time updates, mobile access, and a user-friendly interface. The university's IT department has decided to launch a modernisation project to replace this system with a new, flexible, and scalable platform.

2. The New System – "NexusEnroll":

The new system, "NexusEnroll," must address the following key requirements:

- **Student Self-Service:** Students should be able to browse course catalogues, add/drop classes, view their schedule, and check their academic progress.
- **Faculty Management:** Faculty members need to see their class rosters, submit grades, and request course changes.
- **Administrator Control:** University administrators require tools to manage course offerings, student records, and generate reports on enrolment trends and faculty workload.
- **Scalability:** The system must be able to handle a high volume of simultaneous users, especially during peak enrolment periods.
- **Modern Accessibility:** The system should be accessible via **both** web browsers and mobile applications.

3. Detailed Functional Description of "NexusEnroll"

1. Student Module

The student module is the primary interface for the student body. Its core functions are:

- **Course Catalogue Browse:**
  - Students can search for courses by department, course number, keyword, and instructor.
  - The system must display real-time information for each course, including:
    - Course name and description
    - Instructor name
    - Available seats vs. total capacity
    - Schedule (days, times, location)
    - Prerequisites
  - **Use Case:** A student wants to browse all computer science courses for the upcoming semester that a specific professor teaches.
- **Registration and Enrolment:**
  - Students can add or drop courses from their enrolment list.
  - The system must validate all enrolment requests.
    - **Validation Rules:**
      - Prerequisite checks: The student must have completed all prerequisites for a course.
      - Capacity checks: The course cannot be full.
      - Time conflict checks: The requested course cannot overlap with an already-enrolled course.
  - **Use Case:** A student attempts to enrol for a course. The system must first check if the student has the necessary prerequisites. If they do, they must check if there's available capacity. If both conditions are met, the enrolment is confirmed, and the student's schedule and academic record are updated.

- **Personal Schedule Management:**
    - Students can view their current and past semester schedules.
    - The system should dynamically build and display a calendar-like view of their classes.
- **Academic Progress Tracking:**
    - Students can view a list of completed courses and the grades received.
    - The system should show which courses are still required for their degree program.

2. Faculty Module

The faculty module provides tools for instructors to manage their classes.

- **Class Roster Viewing:**
    - Instructors can view a real-time list of all students currently enrolled in their courses.
    - The roster should include student names, IDs, and contact information.
- **Grade Submission:**
    - Instructors can enter and submit final grades for each student in their courses at the end of the semester.
    - The system must have a process for grade approval (e.g., a "Pending" state before a final "Submitted" state).
    - **Use Case:** A professor submits a batch of grades for a class. The system should process these grades and update the student's academic records. If an error occurs during the process (e.g., an invalid grade is submitted), the system must handle it gracefully and allow the professor to correct it without losing other submitted grades.
- **Course Information Management:**
    - Instructors can submit requests to update course descriptions, add prerequisites, or change course capacity (these requests must be approved by an administrator).

3. Administrator Module

The administrator module is the central control panel for the entire system, with a focus on system-wide management and reporting.

- **Course & Program Management:**
    - Administrators can create, edit, and delete courses.
    - They can define and manage degree programs, including the required courses and credits.
- **Student & Faculty Management:**
    - Administrators can add, edit, and deactivate student and faculty accounts.
    - They can manually override enrolment rules (e.g., force-add a student into a full class).
- **Reporting & Analytics:**
    - The system must generate various reports:
        - Enrolment statistics by department and semester.
        - Faculty workload reports.
        - Course popularity trends.
    - **Use Case:** An administrator needs to generate a report on all courses in the Business school that are currently at over 90% capacity. The system should retrieve this data and present it in a clean, organised format (e.g., a table or spreadsheet).

4. System-Wide Requirements

These requirements apply across all modules and user types.

- **Notification System:**
    - The system must have a notification (e.g. email) mechanism. For example:
        - Students should be notified when a course they are waitlisted for becomes available.
        - Advisors should be notified when one of their advisees drops a critical course.

        - Administrators should be notified of any system-wide errors.
    - **Use Case:** A student drops a course. The notification system must automatically alert any waitlisted students that a spot has opened up. This process should be automated and decoupled from the core enrolment logic.
- **Transaction Management:**
    - All enrolment operations (adding/dropping a course) must be treated as transactions. This means either the entire operation succeeds (e.g., the student is enrolled, the class capacity is updated, and the schedule is modified) or the entire operation fails, leaving no partial state changes.
- **User Interface:**
    - The front-end should be a single-page application (SPA) that communicates with the back-end via APIs. This allows for a responsive user experience.
    - The same back-end services must be usable by both the web application and a future mobile application.

You don't understand anything until you learn it more than one way - Marvin Minsky

[END]