Module:	**Software Design Patterns** Software Architecture and Design Patterns Assessment
Hand-out Date:	19th September 2024
Hand-In Date:	Each Student will submit their document to Canvas and then present their project to me during the last week of lectures on the week ending: 6**th December 2024**
Lecturer:	Andrew Shields (andrew.shields@mtu.ie)

Aim: This coursework aims to assess your knowledge and awareness of design patterns, architecture patterns, and contemporary software development technologies by tackling a real-world software refactoring scenario or developing a system from scratch. In this assignment, you will have the opportunity to customize your scenario, apply various design patterns and architecture patterns, and create a well-documented architectural redesign.

Learning Outcomes:

- Build and design a complex business application that satisfies an architectural design.
- Evaluate the strengths and weaknesses of Design Patterns.
- Apply technical proficiency in Design patterns and modular programming.
- Implementation of the system using a well-designed and justified architecture and frameworks (if appropriate) of your choice.
- Produce a presentation/demonstration to discuss the used technologies and show a working prototype.

Assignment Description:

Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. The architecture of a software system is a metaphor, analogous to the architecture of a building. It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams. Software architecture is about making fundamental structural choices that are costly to change once implemented. Software architecture choices include specific structural options from possibilities in the design of the software.

Assignment Tasks:

1. Customized Scenario Selection or System Development (10%)

- **Task:** Choose or propose your own legacy software system for refactoring or develop a new system from scratch. The system should be sufficiently complex to challenge your understanding of design patterns and architecture patterns.

- **Requirements:** Submit a brief description of the selected system or the new system's concept, explaining why it is suitable for this assignment. Ensure it includes a variety of functional and non-functional requirements.

2. Comprehensive Design Pattern Exploration (15%)

- **Task:** Explore design patterns from different categories, including Creational, Structural, and Behavioural patterns. Additionally, research and explore architectural patterns.
- **Requirements:** Select at least one design pattern from each category and one architecture pattern. Justify your choices, explaining how these patterns can address specific issues within your chosen legacy system or new system. Provide real-world examples of systems that have successfully applied these patterns.

3. Integrated Design and Architecture Pattern Application (20%)

- **Task:** Apply the selected design patterns and architecture patterns to your customized scenario or new system, ensuring that they are effectively integrated within the system.
- **Requirements:** Create UML diagrams that illustrate how each design pattern and the architecture pattern are implemented within the system. Include code samples or snippets to demonstrate the practical application of these patterns. Ensure that your code samples are well-commented for clarity.

4. Detailed Documentation (25%)

- **Task:** Document all your work in a report titled "Software Architecture Overview."
- **Requirements:**
    - Provide a cover page with the title, course, your name, and the date.
    - Include a table of contents for easy navigation.
    - Offer background information on your chosen scenario or new system concept and its significance.
    - Describe in detail the existing structure and design of the application using UML diagrams (for legacy systems) or the initial design (for new systems). Identify at least three specific functional requirements and three non-functional requirements.
    - Justify your design pattern and architecture pattern choices, detailing the pros and cons of each pattern. Include real-world examples where applicable.
    - Present a high-level architecture redesign of the software (for legacy systems) or the finalized architecture (for new systems), highlighting the changes made. Use UML diagrams and code snippets to illustrate key aspects of the design.
    - Testing, Validation, and Code Quality
        - Explanation of testing and validation procedures
        - Identification and removal of code smells
        - Code analysis tools used.
    - Conclusions:

        - Discussing independent informed conclusions based on relevant, critical, and perceptive analysis to arrive at justification of the work completed (hypothesis).
    - Reflections:
        - Clear capacity for an in-depth, comprehensive, and clear self-reflection on the completed work.
        - Consider the learning gained by and through engagement with the work.
        - Considers in depth the reliability and possible error(s) / changes/modifications while comprehensively relating back to the theme and hypothesis/action question.
    - Include a references page with a minimum of six references (two textbooks, two journals, two websites).

5. Presentation/Demonstration (10%)

- **Task:** Prepare and deliver a presentation or demonstration of your refactored system (for legacy systems) or your newly developed system (for new systems). Showcase how the selected design patterns and architecture patterns have improved the system's architecture and functionality.

6. Testing, Validation, and Code Quality (20%)

- **Task:** Implement comprehensive testing and validation procedures to ensure the refactored system (for legacy systems) or the new system (for new systems) meets the specified requirements and functions correctly. Additionally, identify and address code smells and maintain clean code practices throughout the assignment.
- **Requirements:**

  Testing and Validation:

    - Describe your testing approach, including test cases, test data, and expected outcomes. Provide evidence of successful testing and validation. Ensure that your tests cover various aspects of the system, including functionality, performance, and security, as appropriate.

  Code Smells and Clean Code:

    - Utilize code analysis tools to identify code smells within your codebase. Common tools for identifying code smells include:
        - SonarQube
        - PMD
        - Checkstyle
    - Identify and document specific instances of code smells found within your codebase. Discuss the impact of these code smells on maintainability, readability, and overall code quality.
    - Implement corrective actions to address identified code smells, following best practices for clean code. Explain how your refactoring improves code quality and readability.

  - Include comments or annotations in your code to indicate where code smells were identified and subsequently addressed. Explain the changes made and the rationale behind them.

Total Weighting: 100%

Submission Guidelines:

- Submit your assignment according to the instructions provided on Canvas.
- Ensure that your documentation is well-structured, clear, and includes appropriate references.
- Include all necessary diagrams, code snippets, and annotations as outlined in the tasks.

Common Design Patterns:

- Creational Patterns: Singleton, Factory Method, Abstract Factory, Builder, Prototype.
- Structural Patterns: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.
- Behavioural Patterns: Observer, Strategy, Command, State, Chain of Responsibility, Visitor, Template Method, Interpreter, Memento.

Common Software Architecture Patterns:

- Layered Architecture, Microservices, Model-View-Controller (MVC), Event-Driven Architecture, Hexagonal Architecture, Repository Pattern, Service-Oriented Architecture (SOA).

Software Architecture and Design Patterns Assessment Grading Rubric

Criteria	**Excellent (70-100%)**	**Good (50-69%)**	**Satisfactory (30-49%)**	**Poor (0-29%)**
1. Customized Scenario Selection or System Development (10%)	Student has chosen or proposed a system with clear complexity, aligning well with design patterns and architecture considerations.	The selected system or new system concept is suitable for the assignment but lacks some complexity or clarity in alignment with design patterns and architecture.	The chosen scenario lacks complexity, or the justification for selection is weak.	The choice of the system or new system concept is inadequate or unclear, demonstrating a lack of understanding of the assignment requirements.
2. Comprehensive Design Pattern Exploration (15%)	Student explores a variety of design patterns and architectural patterns, selecting at least one from each category. Choices are well-justified with real-world examples.	Design pattern exploration is sufficient, covering different categories, but lacks depth or real-world examples.	Limited exploration of design patterns and architectural patterns. Choices are not well-justified, and real-world examples are missing.	Inadequate exploration of design patterns and architectural patterns. Choices are arbitrary or not relevant to the scenario.
3. Integrated Design and Architecture Pattern Application (20%)	Application of design patterns and architecture patterns is seamlessly integrated into the scenario or new system. UML diagrams and code samples effectively demonstrate practical application.	Integration of design patterns and architecture patterns is present, but there are minor issues with clarity or practical demonstration.	Application of design patterns and architecture patterns is visible, but integration is lacking, and demonstration is not clear.	Poor integration of design patterns and architecture patterns into the scenario or new system. Practical application is unclear.

Criteria	Excellent (70-100%)	Good (50-69%)	Satisfactory (30-49%)	Poor (0-29%)
4. Detailed Documentation (25%)	The report is comprehensive, well-organized, and includes all required sections. Background information, existing structure/design, justification, redesign, testing, and reflections are detailed and insightful.	The report is mostly comprehensive, but some sections lack depth or clarity. Some justification or analysis may be brief.	The report is basic, lacking in-depth analysis in several sections. Some required elements may be missing.	The report is poorly organized, lacks detail, and major sections are incomplete or missing.
5. Presentation/Demonstration (10%)	The presentation effectively showcases the refactored or new system, highlighting improvements due to design patterns and architecture patterns. Clear and engaging delivery.	The presentation showcases the system, but improvements due to design patterns and architecture patterns are not clearly articulated. Delivery is adequate.	The presentation lacks clarity or engagement, and improvements are not effectively communicated. Delivery needs improvement.	The presentation is unclear, lacks engagement, and fails to communicate improvements made to the system.
6. Testing, Validation, and Code Quality (20%)	Comprehensive testing and validation procedures are implemented, covering various aspects of the system. Code smells are identified, addressed, and well-documented. Clean code	Testing and validation are present, but there may be some gaps or lack of coverage. Code smells are identified, but documentation and corrective actions are not	Testing and validation procedures are basic, with notable gaps in coverage. Code smells are identified, but corrective actions lack detail. Clean code practices are not	Testing and validation procedures are inadequate, and major aspects are missing. Code smells are not effectively identified or addressed. Clean code

Criteria	Excellent (70-100%)	Good (50-69%)	Satisfactory (30-49%)	Poor (0-29%)
	practices are followed.	comprehensive. Clean code practices are partially followed.	consistently followed.	practices are ignored.
Total Weighting: 100%				