Pokhara University
Faculty of Science and Technology

Course No.: CMP 340
Course title: **Software Design and Architecture (3-1-2)**
Nature of the course: Theory & Practical

Level: Bachelor

Full marks: 100
Pass marks: 45
Time per period: 1 hour
Total periods: 45
Program: BE

1. Course Description

Software Design and Architecture is an advanced course that builds on fundamental software engineering principles, focusing on designing robust, scalable, and maintainable software systems. Students will explore key concepts such as architectural styles, design patterns, and component-based design, learning how to apply these techniques to real-world projects. The course covers the decision-making process involved in selecting appropriate architectural solutions and navigating trade-offs between functional and non-functional requirements. Through case studies and hands-on projects, students will gain experience in designing software architectures that address complex system needs, ensuring performance, scalability, and security. This course prepares students for roles as software designers and architects in large-scale software development projects.

2. General Objectives

- To equip students with a deep understanding of architectural styles and design patterns to create scalable and maintainable software systems.
- To enable students to make informed architectural decisions by evaluating trade-offs and balancing system performance, security, and flexibility.
- To develop proficiency in applying advanced design principles such as SOLID, DRY, and component-based design for efficient system development.
- To provide practical knowledge of architecture validation methods to ensure that architectural designs meet both functional and non-functional requirements.
- To prepare students to design software for emerging technologies such as distributed systems, microservices, cloud-native applications, and highly available systems.

3. Methods of Instruction

3.1.General instructional Techniques: Lecture, discussion, readings.
3.2.Specific instructional Techniques: Lab works, case study

4. Contents in Detail.

Specific Objectives	Contents
- To understand the fundamental concepts of software design and architecture.	**Unit 1: Introduction to Software Design and Architecture (6 hrs)** 1.1 Overview of software design and architecture 1.2 Design levels: Architectural vs. detailed design 1.3 Design principles (modularity, abstraction, separation of concerns) 1.4 The role of the software architect 1.5 Key design goals: Performance, scalability, maintainability 1.6 Relationship between software design and software architecture 1.7 Object Oriented Software Development: Unified Software Development Process
- To explore various architectural styles and patterns and their impact on system quality. - To apply design patterns to solve common design challenges.	**Unit 2: Architectural Styles and Patterns (7 hrs)** 2.1 Common architectural styles (Layered, Client-server, Microservices, Event-driven, SOA) 2.2 Architectural patterns (MVC, Broker, Pipe and Filter) 2.3 Design patterns (Singleton, Factory, Observer, Strategy) 2.4 Choosing appropriate architectural styles and patterns based on system requirements 2.5 Impact of architecture on non-functional requirements (performance, scalability, security)
- To apply SOLID principles and best practices for designing modular, maintainable software. - To understand the importance of coupling, cohesion, and design for scalability.	**Unit 3: Software Design Principles and Best Practices (7 hrs)** 3.1 SOLID principles and their application 3.2 DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid) 3.3 Coupling and cohesion 3.4 Reusability, modularity, and maintainability in software design 3.5 Design for change and scalability 3.6 Separation of concerns and information hiding 3.7 Case Study: Implementation of SOLID Principles in the Development.

- To design reusable and maintainable software components and manage component lifecycles. - To understand component composition, integration, and communication in large-scale systems.	**Unit 4: Component-Based Software Design (6 hrs)** 4.1 Introduction to component-based development 4.2 Components, interfaces, and contracts 4.3 Component composition, integration, and communication 4.4 Designing reusable and maintainable components 4.5 Component lifecycle management and versioning 4.6 Component-based design in large-scale systems
- To evaluate architectural trade-offs and balance functional and non-functional requirements.	**Unit 5: Architectural Decision-Making and Trade-offs (6 hrs)** 5.1 Architectural decision-making process 5.2 Balancing functional and non-functional requirements 5.3 Trade-offs in architecture (e.g., performance vs. scalability, security vs. flexibility) 5.4 Risk analysis and mitigation strategies in architecture 5.5 Justifying architectural decisions and documenting trade-offs 5.6 Case study: A system developed using Microservices Architecture
- To understand and apply methods for evaluating and validating software architectures.	**Unit 6: Software Architecture Evaluation and Validation (6 hrs)** 6.1 Architecture evaluation methods (ATAM, CBAM) 6.2 Scenario-based architecture validation 6.3 Architecture reviews and continuous validation 6.4 Assessing architecture against system qualities (performance, security, usability) 6.5 Tools for architecture evaluation and validation 6.6 Architecture evaluation in agile and DevOps environments

- To design for distributed systems, microservices, and cloud-native architecture. - To understand advanced design strategies for concurrency, fault tolerance, and high availability.	**Unit 7: Advanced Topics in Software Design and Architecture (7 hrs)** 7.1 Designing for distributed systems and cloud-native architecture 7.2 Microservices and serverless architecture 7.3 Domain-Driven Design (DDD) 7.4 Designing for concurrency and parallelism 7.5 Fault tolerance and high availability

5. List of Tutorials:

The following tutorial activities of 15 hours per group of maximum 24 students should be conducted to cover all the required contents of this course.

S.N.	Tutorials
1	Case study on design Patterns Workshop
2	Case study on Domain-Driven Design (DDD)
3	Discussion on Microservices Architecture Simulation:
4	Case study on ATAM Evaluation
5	Discussion on Scenario-Based Architecture Evaluation:
6	Discussion on Clean Architecture Coding
7	Case study on Event-Driven Architecture Design

6.List of Practical

The following Practical works of 30 hours per group of maximum 24 students should be conducted to cover all the required contents of this course.

7. Evaluation system and Students' Responsibilities

Internal Evaluation

The internal evaluation of a student may consist of assignments, attendance, test-exams, term-exams, lab reports and projects etc. The tabular presentation of the internal evaluation is as follows:

S.N.	Practical
1	Students will explore and apply common design patterns such as Singleton , Factory Observer, Strategy by working in pairs to implement a simple application..
2	Students will model a given domain using Domain-Driven Design principles. They will identify key entities and relationships, create a domain model diagram, and explain their design choices to the class.
3	Students will design a microservices architecture for a specified use case, breaking down the system into services and defining their interactions. They will present their architecture and discuss deployment strategies and potential challenges.
4	Using the Architecture Tradeoff Analysis Method (ATAM), students will evaluate a provided software architecture based on quality attributes and trade-offs. They will complete an evaluation checklist and present their findings and recommendations.
5	Students will create and analyze scenarios to assess the effectiveness of a given software architecture. They will determine how well the architecture meets specific quality attributes and present their evaluation results.
6	Students will refactor a provided codebase to adhere to Clean Architecture principles, focusing on separation of concerns and maintainability. They will discuss their refactoring approach and the improvements made.
7	Students will design an event-driven system for a given problem, identifying events, event handlers, and communication patterns. They will present their system design and address potential challenges and scalability issues.

External Evaluation	Marks	Internal Evaluation	Weight	Marks
Semester-End examination	50	Attendance	10%	50
		Tutorial/Case Studies	40%	
		Term exam	50%	
	50	Internal Final	100%	50
Full Marks 50+50= 100				

Student Responsibilities:

Each student must secure at least 45% marks in internal evaluation with 80% attendance in the class in order to appear in the Semester-End Examination. Failing to get such score will be given NOT QUALIFIED (NQ) and the student will not be eligible to appear the Semester-End Examination. Students are advised to attend all the classes and complete all the assignments within the specified time period. If a student does not attend the class(es), it is his/her sole responsibility to cover the topic(s) taught during the period. If a student fails to attend a formal exam, test, etc. there won't be any provision for re-exam.

7. Prescribed Books and References

Books and Standards

1. Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley. ISBN: 978-0321815736
2. Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley. ISBN: 978-0321125217
3. Kruchten, P. (2004). *The rational unified process: An introduction* (3rd ed.). Addison-Wesley. ISBN: 978-0321197700
4. IEEE Standards Association. (2017). *IEEE Std 1471-2000 - IEEE recommended practice for architectural description of software-intensive systems*.
5. ISO/IEC/IEEE 42010:2011 - Systems and software engineering – Architecture description.

Online Resources

1. Software Engineering Body of Knowledge (SWEBOK). (n.d.). *SWEBOK Guide*.
2. IEEE Xplore Digital Library. *IEEE software engineering standards*.
3. Kazman, R., Bass, L., & Abowd, G. (2000). Scenario-based analysis of software architecture. *ACM SIGSOFT Software Engineering Notes, 25*(1), 37-48.

Supplement Materials

Books

1. Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media. ISBN: 978-1449373320
2. Martin, R. C. (2017). *Clean architecture: A craftsman's guide to software structure and design*. Prentice Hall. ISBN: 978-0134494166