**UNIT-I: Introduction to Object-Oriented Programming (OOP)**

* **Overview of Software Engineering**: Discusses the principles, importance, and goals of software engineering.
* **OOP Concepts**: Introduces fundamental OOP principles:
  + **Classes and Objects**: Fundamental building blocks.
  + **Inheritance**: Code reuse and hierarchy modeling.
  + **Polymorphism**: Unified interfaces for varied implementations.
* **Unified Modeling Language (UML)**: Basics of visualizing, specifying, and documenting system designs.
* **Software Development Life Cycle (SDLC)**: Phases, including planning, analysis, design, implementation, and maintenance.

**UNIT-II: Requirements Analysis and Design**

* **Requirements Analysis**: Gathering and documenting functional and non-functional requirements.
* **Use Cases and Scenarios**: Defining system functionalities and interactions.
* **Object-Oriented Analysis and Design (OOAD)**: Identifying objects, classes, and relationships.
* **Design Patterns**: Reusable solutions for common design problems.
* **UML Modelling Techniques**:
  + **Class Diagrams**: Static structure of systems.
  + **Sequence Diagrams**: Interaction flows over time.
  + **State Machine Diagrams**: State transitions of objects.
  + **Activity Diagrams**: Workflows and system activities.

**UNIT-III: Software Construction and Testing**

* **Software Construction Basics**: Writing, debugging, and maintaining code.
* **Object-Oriented Design Principles**: SOLID principles and best practices.
* **Programming Languages**: Exploration of OOP in Java, C++, and Python.
* **Software Testing Basics**:
  + **Unit Testing**: Testing individual components.
  + **Integration Testing**: Ensuring modules work together.
  + **System Testing**: Verifying the complete system.
* **Test-Driven Development (TDD)**: Writing tests before code.

**UNIT-IV: Software Maintenance and Evolution**

* **Software Maintenance**: Corrective, adaptive, and perfective updates.
* **Refactoring Techniques**: Improving code quality without altering functionality.
* **Software Version Control**: Tracking and managing changes (e.g., Git).
* **Code Review and Inspection**: Ensuring code quality and standards.
* **Software Evolution and Reengineering**: Adapting and modernizing software.

**UNIT-V: Advanced Topics in OOSE**

* **Model-Driven Engineering (MDE)**: Automated generation of software artifacts from models.
* **Aspect-Oriented Programming (AOP)**: Modularity for cross-cutting concerns.
* **Component-Based Software Engineering (CBSE)**: Building systems using reusable components.
* **Service-Oriented Architecture (SOA)**: Designing software as loosely coupled services.
* **Agile and Scrum**: Iterative development with emphasis on collaboration and flexibility.

**Learning Outcomes**

Upon completing this course, students should be able to:

1. Understand and apply OOP principles.
2. Perform UML modeling for real-world systems.
3. Analyze requirements and design object-oriented systems.
4. Write, test, and maintain software using modern techniques.
5. Apply advanced concepts like MDE, AOP, and Agile in software projects.