System Modeling & UML

Lecture A1 — An Introduction to Object-Oriented Modeling

Prof. Ing. Lelio Campanile

Dipartimento di Matematica e Fisica
Università degli Studi della Campania Luigi Vanvitelli
Corso di Laurea Magistrale in Data Science

A.A. 2025/2026

Based on: I. Sommerville — *Software Engineering* | Seidl et al. — *UML @ Classroom* (Springer, 2015)

Topics covered

Part 1 — Foundations

- Why modeling?
- System modeling and perspectives
- Static vs. dynamic modeling
- How models are used

Part 2 — UML in Practice

- UML diagram families
- The five diagrams of this course
- Levels of abstraction
- UML in the development process
- Running example

Part 1 — Foundations

Why model? What can we model?

Why modeling?

Core motivation

"Natural language is not necessarily a good choice — it is imprecise and ambiguous. What you need is a model."

Seidl et al., *UML @ Classroom*, 2015

Communication across roles

Data scientists, developers, clients and managers rarely share the same vocabulary. Models provide a neutral, precise *lingua franca*.

Abstraction & clarity

A model highlights what matters and hides irrelevant detail — like an architectural blueprint vs. the real building.

Early error detection

Inconsistencies caught in a diagram are orders of magnitude cheaper to fix than bugs found in production code.

System modeling

Definition

System modeling is the process of developing abstract models of a system, each presenting a different *view* or *perspective*. Today it almost always means graphical notation based on **UML**.

Sommerville, *Software Engineering*

Models of the *existing* system

- Used during requirements engineering
- Clarify what the system currently does
- Basis for discussing strengths and weaknesses

Models of the *new* system

- Explain proposed requirements to stakeholders
- Support design discussions
- Document the system for implementation
- Enable code generation in MDE/MDA contexts

System perspectives

Four complementary views (Sommerville)

No single model is sufficient. Each perspective captures a different aspect of system complexity, and they are used together.

External

Context and environment. What lies *outside* the system; how the boundary is defined.

Interaction

How the system communicates with users and other systems; how internal components interact.

Structural

Organization of the system and the data it processes.

$\Rightarrow$ **static view**

Behavioral

Dynamic behavior: how the system responds to events over time.

$\Rightarrow$ **dynamic view**

How models are used

Key principle

The required level of detail and precision depends on the **purpose** and **audience** of the model.

Facilitating discussion

Incomplete or imprecise models are *acceptable*. The model exists to support dialogue, not to be archived.

Documenting a system

Models should accurately represent the system but need not be complete. Focus: architectural clarity for future maintainers.

Driving implementation

Models must be both *correct* and *complete*. Used in Model-Driven Engineering: the model is the source of truth.

Static modeling vs. Dynamic modeling

Two fundamental and complementary ways to describe a software system

Static modeling

- Describes the **structure** of the system
- Classes, objects, and their relationships
- Does not change over time within the model
- *Answers: What entities exist? How are they related?*

Diagrams in this course:

✔ Use Case Diagram
✔ Class Diagram

Dynamic modeling

- Describes the **behavior** of the system over time
- Interactions, sequences, flows, state transitions
- Captures how the system responds to events
- *Answers: What happens? In what order?*

Diagrams in this course:

✔ Sequence Diagram
✔ Activity Diagram
✔ State Machine Diagram

Part 2 — UML in Practice

Diagram families, abstraction levels, development process

UML diagram families

UML defines **14 diagram types** in two families. In this course we cover the **5 most widely used** in practice.

Structure Diagrams (static)

- ✔ **Class Diagram**
- ✔ **Use Case Diagram**
- Object Diagram
- Package Diagram
- Component Diagram
- Deployment Diagram
- ...

Behavior Diagrams (dynamic)

- ✔ **Sequence Diagram**
- ✔ **Activity Diagram**
- ✔ **State Machine Diagram**

Interaction diagrams:

- Communication Diagram
- ...

✔/✔ = covered in this course

The five diagrams of this course

Diagram	Type	Key question it answers
Use Case	Static	Who interacts with the system, and what can they do?
Class	Static	What are the main entities and how are they structured?
Sequence	Dynamic	In what order do objects interact to realize a scenario?
Activity	Dynamic	What is the flow of a process, algorithm, or workflow?
State Machine	Dynamic	How does the system's state change in response to events?

Levels of abstraction in UML

Key principle

The **same diagram type** can be drawn at different levels of detail. Choosing the level is a *deliberate design decision* based on audience and purpose.

1 — Conceptual

Class/entity names *only*. No attributes, no operations.

Use: early requirements discussions with stakeholders.

2 — Specification

Attributes and operations visible, with types.

Use: design reviews and API/interface definition.

3 — Implementation

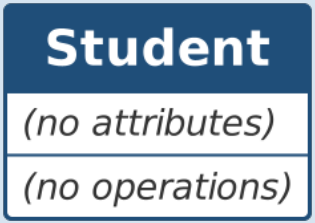
Full detail: visibility, types, multiplicities, properties.

Use: blueprint for code generation.

Levels of abstraction — Class Diagram example

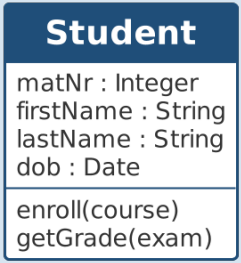
The class `Student` in the Student Administration System at three levels

Level 1 — Conceptual



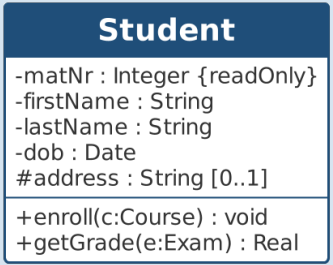
Used in early requirements discussions with stakeholders.

Level 2 — Specification



Used for design reviews and API definition.

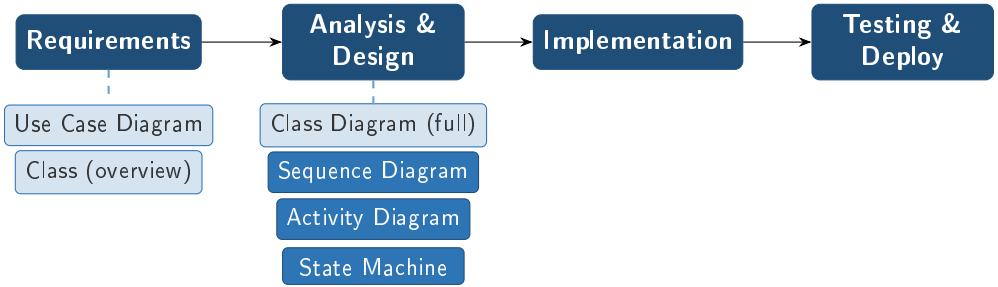
Level 3 — Implementation



Blueprint for code generation.

UML in the software development process

Each diagram type has a natural home in the development lifecycle



Note: Use Case and Class Diagrams also serve as **documentation** across all phases of development.

Running example: Student Administration System

Used throughout Modules A and B to illustrate all five diagram types

System

A university system for managing students, courses, exams, and administrative procedures.

Actors

- Professor
- Student
- Admin Staff
- E-Mail Server (non-human actor)

Main functionalities (use cases)

- Enroll in course
- Query student data
- Announce exam
- Issue grade certificate
- Reserve lecture hall

What we will model

- Use Case Diagram
- Class Diagram (3 levels)
- Sequence Diagram
- Activity Diagram

Key points

Models are abstractions

A model captures what matters and hides irrelevant detail. It is a shared language between people with different backgrounds.

Dynamic diagrams describe behavior

Sequence, Activity, and State Machine Diagrams capture *how the system behaves over time* and in response to events.

Static diagrams describe structure

Use Case and Class Diagrams capture *what the system is* and *how it is organized* — they do not evolve over time within the model.

Detail level is a deliberate choice

The same diagram can be a sketch for a stakeholder meeting or a blueprint for code generation. Choose based on **audience** and **purpose**.

What's next

Lecture A2 — Use Case Diagram

Modeling system functionality from the user's perspective.

Seidl et al., Ch. 6

Lecture A3 — Class Diagram

Modeling the static structure: from conceptual overview to implementation-level detail.

Seidl et al., Ch. 3

Module B — Behavioral Modeling

- Sequence Diagram *(Ch. 4)*
- Activity Diagram *(Ch. 5)*
- State Machine *(Ch. 7)*

Modeling tools

draw.io — web-based, free
PlantUML — text-based, Git-versionable
StarUML — desktop, UML 2.x compliant

Thank you

Prof. Ing. Lelio Campanile

lelio.campanile@unicampania.it

Dipartimento di Matematica e Fisica
Università degli Studi della Campania Luigi Vanvitelli

Elements of Software Engineering and Information Systems

Corso di Laurea Magistrale in Data Science