

# 1 Lecture 2: Unified Modeling Language (UML)

**Course:** Software Engineering and Systems Analysis

**Topic:** UML as a Modeling Language

**Recommended duration:** 90 minutes

## 1.1 Learning objectives

Students should be able to define UML, explain why different diagram types are needed, select an appropriate diagram for a modeling question, read basic UML notation, and apply a disciplined modeling workflow.

## 1.2 1. What UML is

The **Unified Modeling Language (UML)** is a standardized visual language for specifying, visualizing, constructing, and documenting software-intensive systems. UML is a language, not a development methodology. It defines concepts and notation, but it does not prescribe whether a team should use Scrum, the waterfall model, or another process.

A model is a purposeful abstraction. It leaves out details that are not relevant to the question being investigated. Therefore, the same system may have several valid UML diagrams, each emphasizing a different aspect of the system.

## 1.3 2. Why multiple diagrams are necessary

A requirements analyst may need to ask what users want to achieve. A use case diagram addresses that question. A class diagram asks what concepts and relationships exist. A sequence diagram asks how objects collaborate over time. A state machine diagram asks how an object changes in response to events.

No single diagram can communicate all of these dimensions effectively. Multiple diagrams are useful when each diagram has a clear purpose and the diagrams do not contradict one another.

## 1.4 3. Main UML diagram families

UML diagrams are commonly grouped into **structural diagrams** and **behavioral diagrams**. Structural diagrams describe what exists. They include class, component, package, deployment, object, and composite structure diagrams.

Behavioral diagrams describe what happens. They include use case, activity, state machine, sequence, communication, interaction overview, and timing diagrams. The exact classification may vary slightly by UML edition, but the underlying distinction remains useful.

The following compact comparison helps select a diagram:

| Question                                    | Useful diagram        |
|---------------------------------------------|-----------------------|
| What goals do external users have?          | Use case diagram      |
| What concepts and relationships exist?      | Class diagram         |
| What workflow or business process occurs?   | Activity diagram      |
| How do objects exchange messages over time? | Sequence diagram      |
| How does one object react to events?        | State machine diagram |

| Question                               | Useful diagram     |
|----------------------------------------|--------------------|
| Where are software artifacts deployed? | Deployment diagram |

## 1.5 4. Basic notation principles

UML notation uses named elements, graphical connectors, and constraints. A name should express the model element's meaning rather than its implementation. A connector should have a defined semantic interpretation. If a line is only decorative, it should be removed.

A **stereotype** extends UML vocabulary for a specific domain. It appears between guillemets, such as «interface» or «service». A **constraint** expresses a rule that is not fully represented by the basic notation. Constraints are often written in braces, for example {ordered} or {unique}.

UML also supports notes. Notes are valuable for assumptions and business rules, but they should explain the model rather than replace missing structure.

## 1.6 5. Modeling workflow

Begin with the question the model must answer. Define the system boundary and intended audience. Select the smallest diagram type that can answer the question. Identify the principal elements, name them consistently, and add relationships only when they carry meaning.

Next, validate the diagram with stakeholders or with concrete scenarios. Check whether the model explains the scenario without requiring undocumented assumptions. Revise the model when the requirements change. A diagram is a communication artifact, not a permanent truth.

## 1.7 6. Consistency across diagrams

A UML model is stronger when its diagrams use the same vocabulary. If the use case diagram names an external role “Customer,” the class and sequence diagrams should not call the same concept “Client” without an explicit reason.

Traceability is also important. A use case can be realized by one or more interaction diagrams. A class can participate in the operations needed by a use case. A deployment diagram can show where the components that support those interactions run.

Consistency does not mean that every element must appear in every diagram. It means that shared elements have compatible meanings and constraints.

## 1.8 7. Example: appointment system

For a clinic appointment system, a use case diagram may show Patient, Receptionist, and Doctor interacting with the system. A class diagram may show Patient, Appointment, Doctor, and Prescription. A sequence diagram for “Book appointment” may show the patient interface sending a request to an appointment service, which checks availability and persists the appointment.

Each diagram answers a different question. Combining all of these elements into one diagram would produce a dense image that is difficult to review.

## 1.9 8. Levels of abstraction

At the conceptual level, a class diagram may contain Appointment and Doctor without database keys or framework classes. At the design level, it may add interfaces, services, repositories, and data types. At the implementation level, it may include language-specific details.

The correct level depends on the decision being made. Adding implementation details too early can hide domain problems. Omitting necessary design details later can prevent developers from understanding the intended architecture.

## 1.10 9. Common modeling problems

A frequent problem is diagram-first modeling: the author chooses a visually attractive notation before deciding what question the model should answer. Another problem is semantic overloading, in which one symbol is expected to represent a user, a database table, and a software component simultaneously.

UML should not be used as decoration. A diagram is successful when a reader can make a better decision after inspecting it. If a diagram does not support communication, analysis, design, or documentation, its maintenance cost may not be justified.

## 1.11 10. Practice activity

Choose an e-learning platform. Create a short modeling portfolio containing one use case diagram, one class diagram, and one sequence diagram for “submit assignment.” Write one paragraph explaining the question answered by each diagram and identify one vocabulary term that must remain consistent across all three.

## 1.12 Summary

UML is a standardized language for modeling different views of a system. The central discipline is to choose a diagram based on the question, control abstraction, and maintain consistency across views. UML becomes valuable when it improves shared understanding and supports concrete engineering decisions.

## 1.13 References