Software Engineering Essentials

# Analysis

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

# Learning Goals

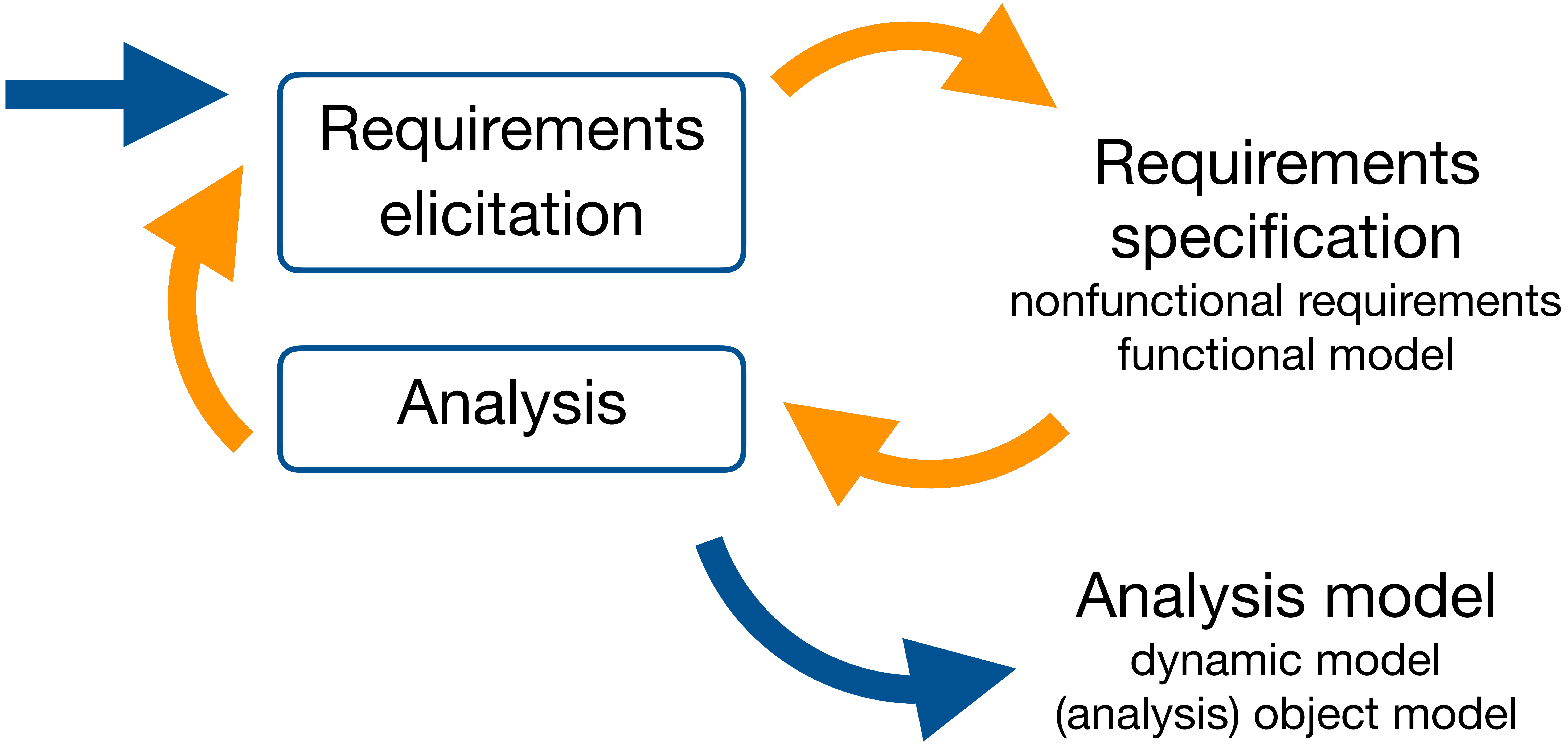
1) Understand the term analysis

2) Understand three different concepts of analysis

3) Apply three different major analysis activities

# Purpose of Analysis

Analysis focuses on producing a model of the system, called the *analysis model*, which is *correct, complete, consistent, and verifiable*.

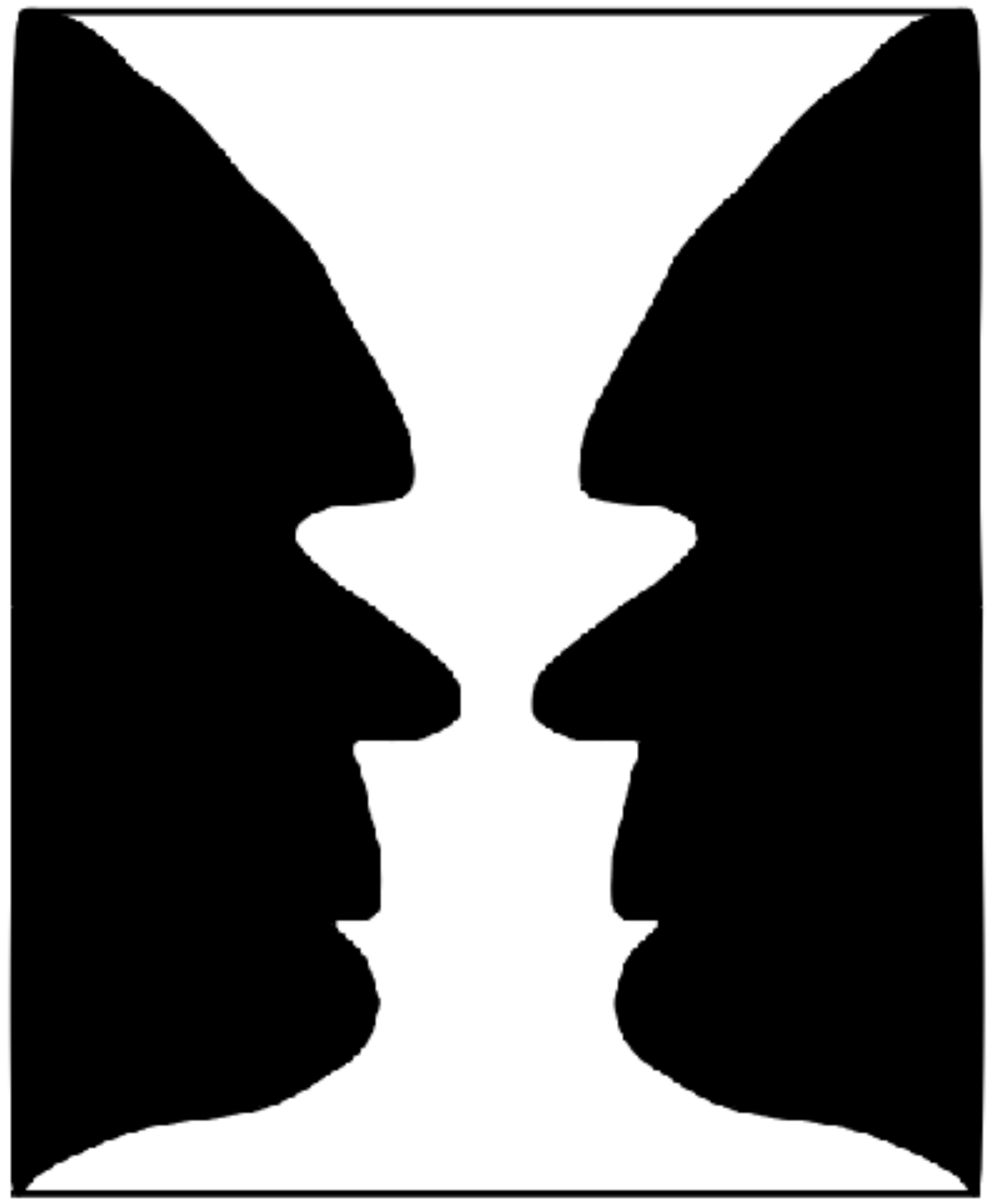


Requirements elicitation

Analysis

Requirements specification
nonfunctional requirements
functional model

Analysis model
dynamic model
(analysis) object model



An optical illusion

Ambiguity in drawings; we need to be consistent in our models

# Formalization during analysis

**Formalization** helps identify areas of ambiguity as well as inconsistencies and omissions

We can deal with ambiguity by using standardized **notations** instead of **drawings**

For Analysis we use the following **UML** diagram types:

use case diagram, class diagram, state machine diagram and sequence diagram

not covered in this course

not covered in this course

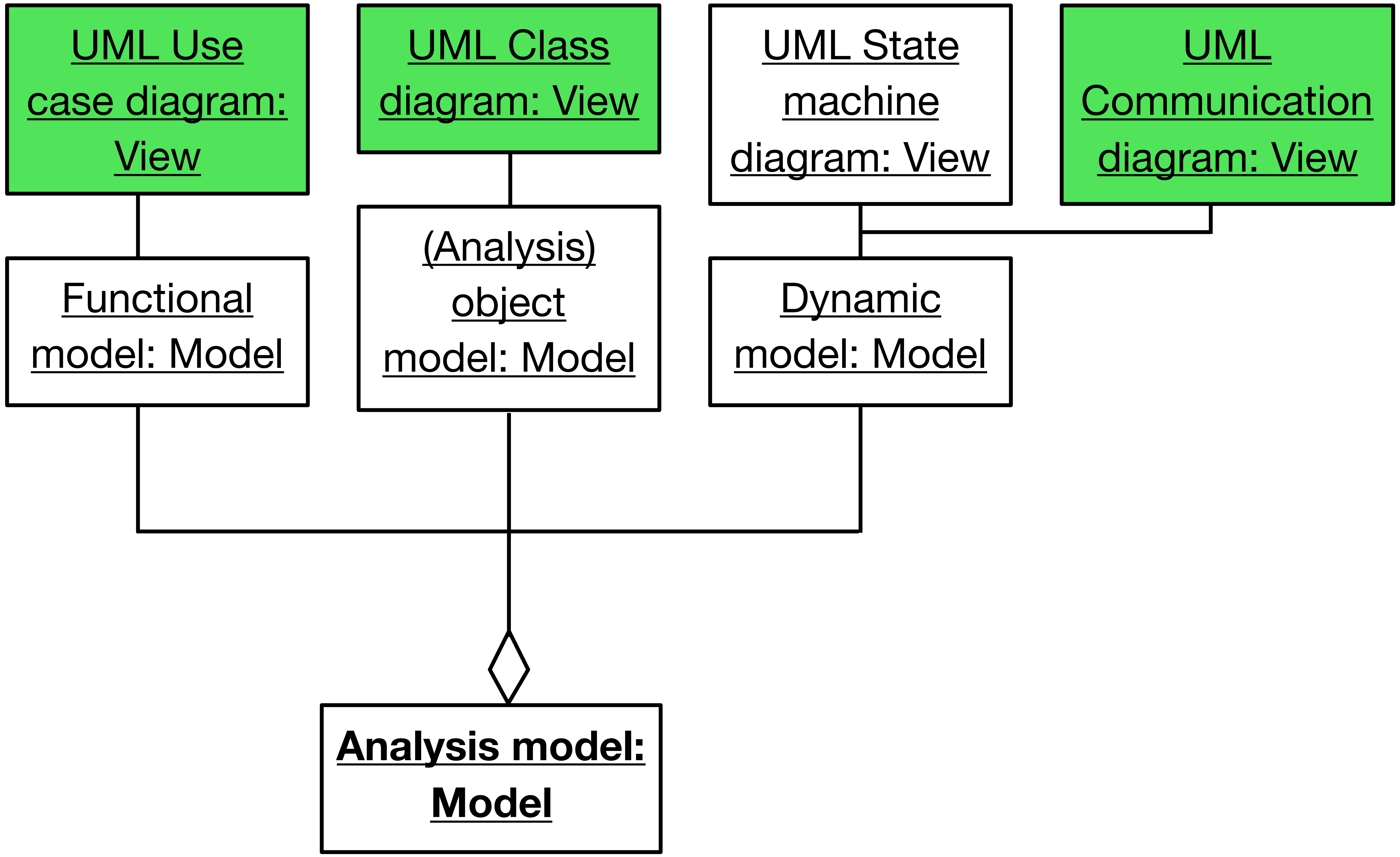
# Analysis concepts overview

**Analysis model:** the functional, (analysis) object and dynamic model of a system to be developed

**Entity, boundary and control objects:** objects can be divided into three major categories describing their use inside the system

**Generalization and specialization:** hierarchies can be detected in two different ways to adopt OOP principles like inheritance/polymorphism and abstraction

# Analysis model

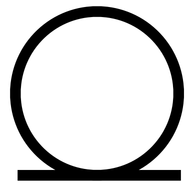

UML Use case diagram: View
UML Class diagram: View
UML State machine diagram: View
UML Communication diagram: View
Functional model: Model
(Analysis) object model: Model
Dynamic model: Model
Analysis model: Model

# Entity, boundary and control objects

The (analysis) object model consists of entity, boundary, and control objects [JA99]

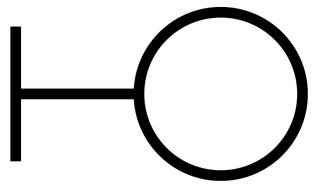
**Entity objects** represent the persistent information tracked by the system.
(e.g. a student object with its containing attributes firstName and LastName)

UML notation:



**Boundary objects** represent the interactions between the actors and the system.
(e.g. a view with buttons and icons)

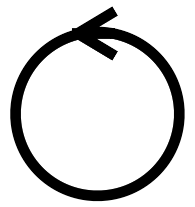
UML notation:



**Control objects** are in charge of realizing use cases.

(e.g. a CourseController which handles the registration of students to courses)

UML notation:



[JA99] I. Jacobson, G. Booch, & J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Reading, MA, 1999

# Generalization and specialization

## Generalization

- Identifies abstract concepts from lower-level ones
- Noticing common features among different concepts, we create an abstract concept

  Common speech: “From low level to high level”

## Specialization

- Identifies specialized concepts from higher level ones
- is the activity that identifies more specific concepts from a high-level one

  Common speech: “From high level to low level”

# Analysis activities overview

1) Identify entity, boundary and control objects

2) Identify associations, aggregates and attributes

3) Model inheritance relationships

# 1) Identify entity, boundary and control objects

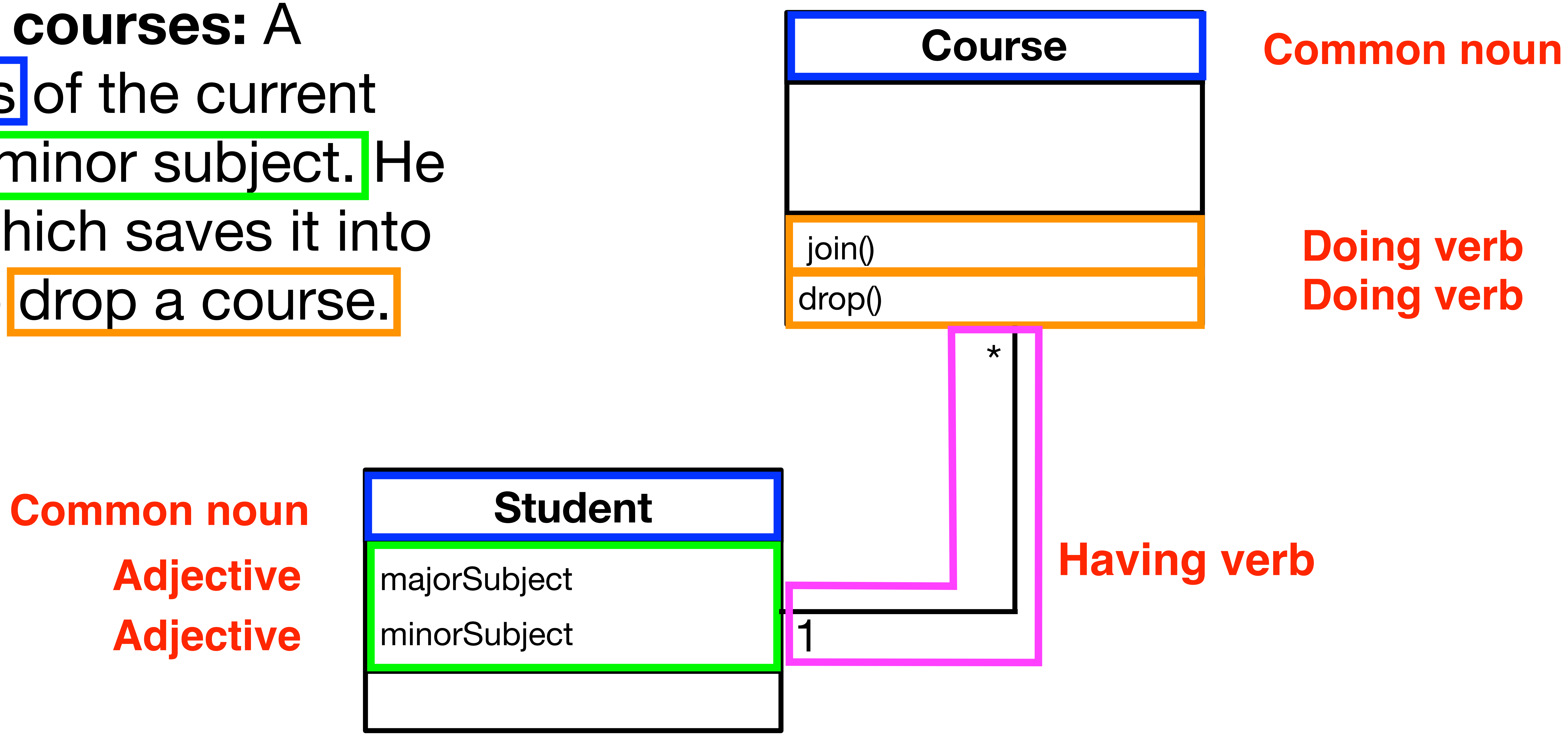
Abbott’s heuristics for mapping parts of speech to model components [AB83]

| **Part of speech** | **Model component** | **Example** |
|---|---|---|
| Proper noun | Instance | Alice |
| Common noun | Class | Student |
| Doing verb | Operation | Creates, submits, selects |
| Being verb | Inheritance | Is a kind of, is one of either |
| Having verb | Aggregation | Has, consists of, includes |
| Modal verb | Constraints | Must be |
| Adjective | Attribute | Incident description |

[AB83] R. Abbott, “Program design by informal English descriptions,” Communications of the ACM, Vol. 26, No. 11, 1983.

# Example Abbott’s technique applied

**FR1: Search for available courses:** A student can see all courses of the current semester in his major and minor subject. He is able to join the course which saves it into his course list. He can also drop a course.



# 2a) Identify associations

An **association** shows a relationship between two or more classes

Associations have the following properties: name, role, multiplicity

The following heuristics can be used to identify associations:

- Examine verb phrases
- Name associations and roles precisely
- Eliminate any association that can be derived from other associations
- Too many associations make a model unreadable

# 2b) Identify aggregates

**Aggregations** are special types of associations denoting a whole–part relationship

There are two types of aggregation:

**1) Composition aggregation:** UML notation: ——◆

➡ The existence of the parts depends on the whole.

**2) Shared aggregation:** UML notation: ——◇

➡ The whole and the part can exist independently.

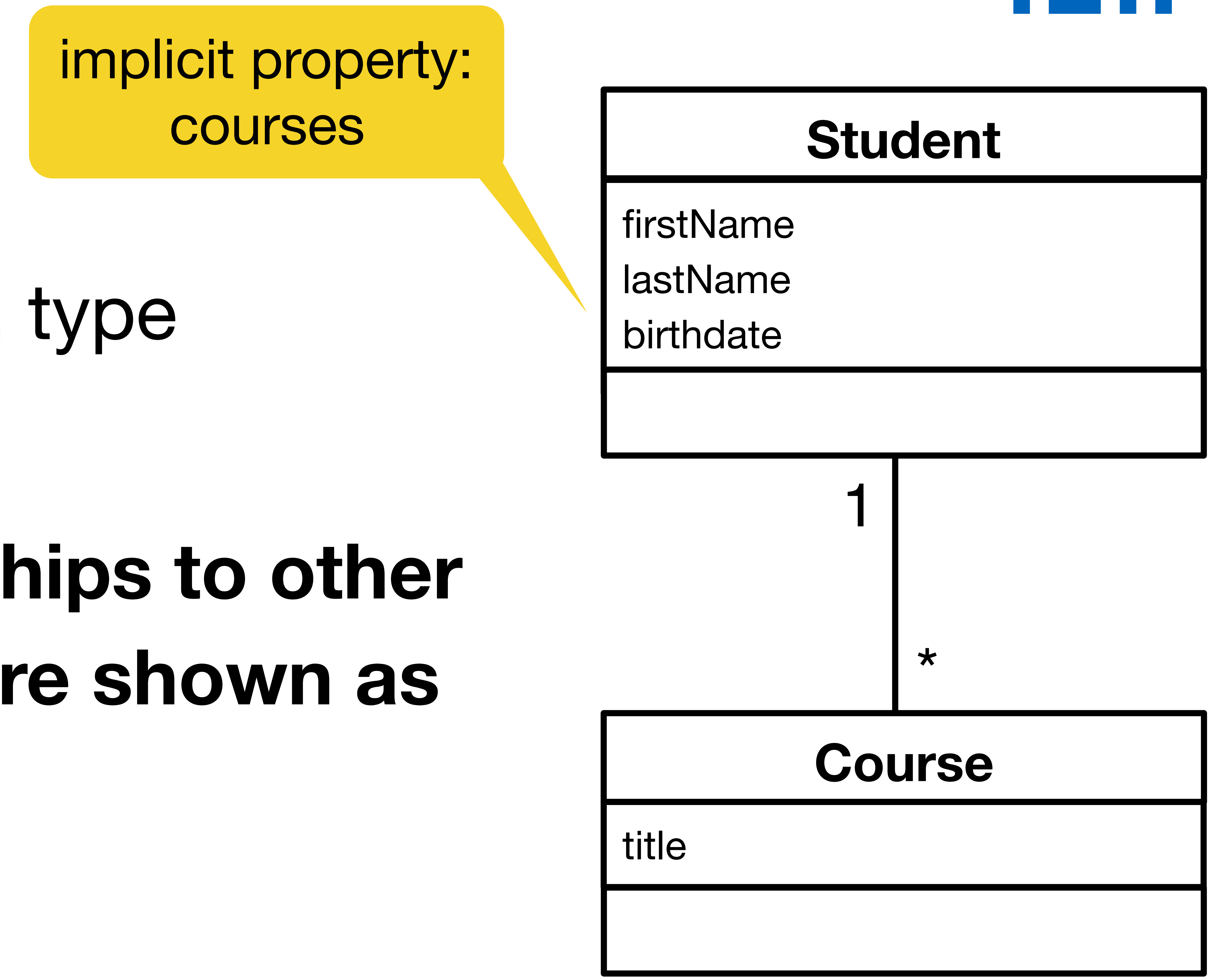
# 2c) Identify attributes

**Attributes** are properties of individual objects.

Attributes have the following properties: name, type

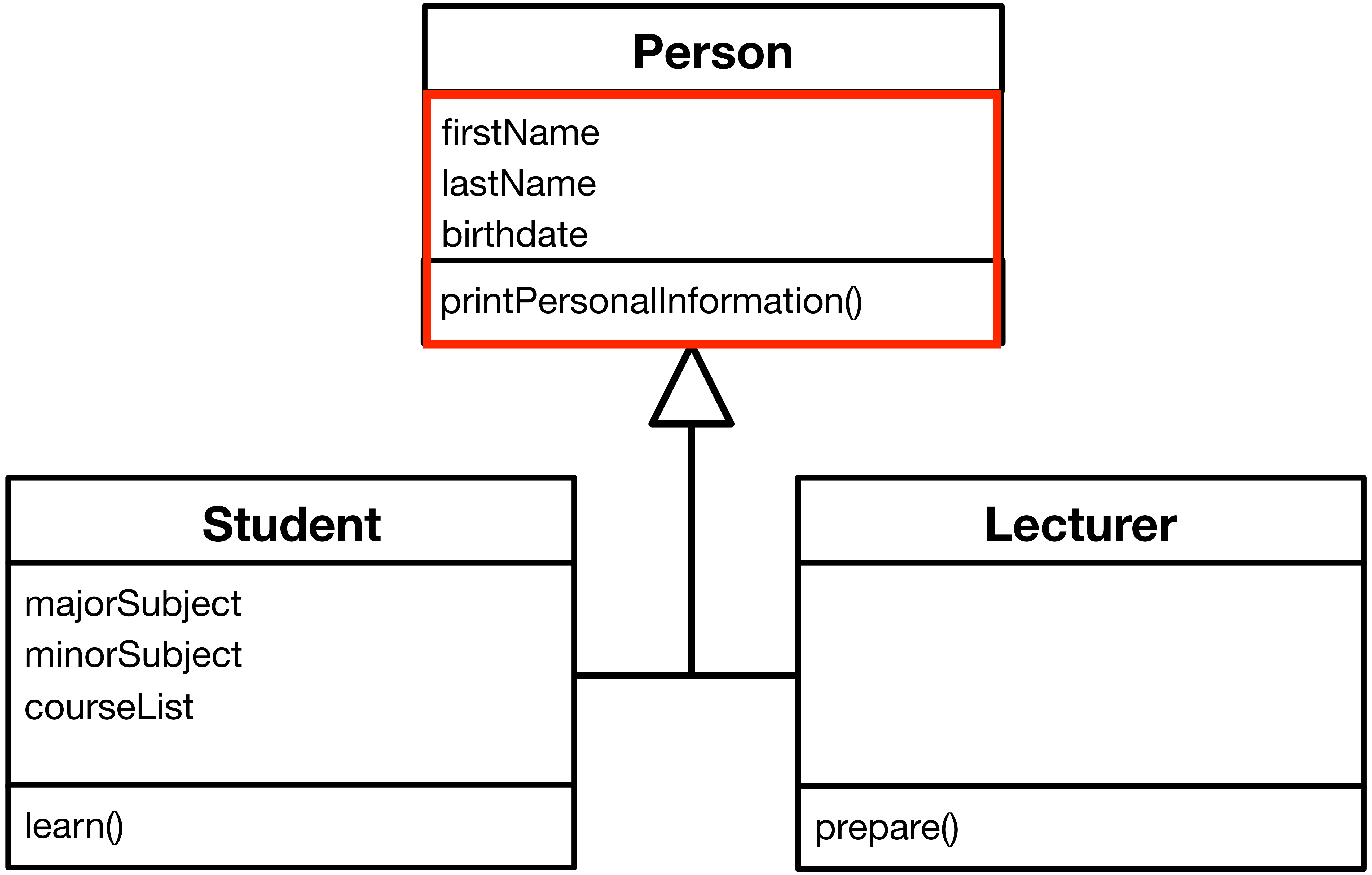
**Properties that are represented by relationships to other objects are not shown as attributes - they are shown as associations**

Heuristic for identifying attributes:
Start with associations and then insert attributes



# 3) Model inheritance relationships

- Generalization is used to eliminate redundancy from the (analysis) object model
- If two or more classes share attributes or behavior, the similarities are consolidated into a superclass



# Summary of analysis

**Analysis highlights key aspects of a system:**

Functional model + (analysis) object model + dynamic model = analysis model

**Entity, boundary and control objects categorize objects:**

- Entity objects are (persistently) stored
- Boundary objects interact with users
- Control objects handle functionality (use cases) of the system

**Generalization and specialization:**

Reusing existing structure and behavior with either a “top-down” or “bottom-up” approach

Software Engineering Essentials

# Analysis

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics