

Part II – Black-Box Composition Systems

10. Business Components in a Component-Based Development Process

1. Business component model of the Cheesman/Daniels process
2. Identifying business components

Prof. Dr. Uwe Aßmann
Technische Universität
Dresden

Institut für Software- und
Multimediatechnik

<http://st.inf.tu-dresden.de>

11-0.2, 27.04.11



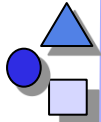


Literature

- ▶ J. Cheesman, J. Daniels. UML Components. Addison-Wesley.
- ▶ R. Prieto-Diaz. Implementing Faceted Classification for Software Reuse. CACM May 1991, vol 34(5). ACM.
<http://doi.acm.org/10.1145/103167.103176>

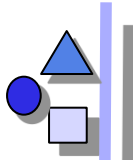
10.1 Business Component Model of the Cheesman-Daniels Process





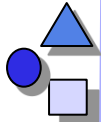
Goals of the Chessman-Daniels Process

- ▶ Bring together domain modelling with use case modelling (functional requirements)
- ▶ Find out business objects (large objects, subjects) of application
- ▶ Group business objects to components for change-oriented design and reuse
- ▶ Specify contracts for the components



Business Objects are Complex Object (Subjects)

- ▶ A **business object** is complex object (subject) with a coarse-grain, natural type of the domain model (business model)
 - which lives on its own (*natural type*)
 - exists independent of context and collaborators
 - which does not depend on other types (*independent type*)
 - Hotel vs. HotelRoom
 - Car vs. Screw or Motor
 - We call types that depend on others *dependent types*.
- ▶ Usually, business objects are large units
 - They can consist of thousands of smaller objects of dependent types (part-of relation)
 - They can play many *roles* with *context-based types*



Business Component Model

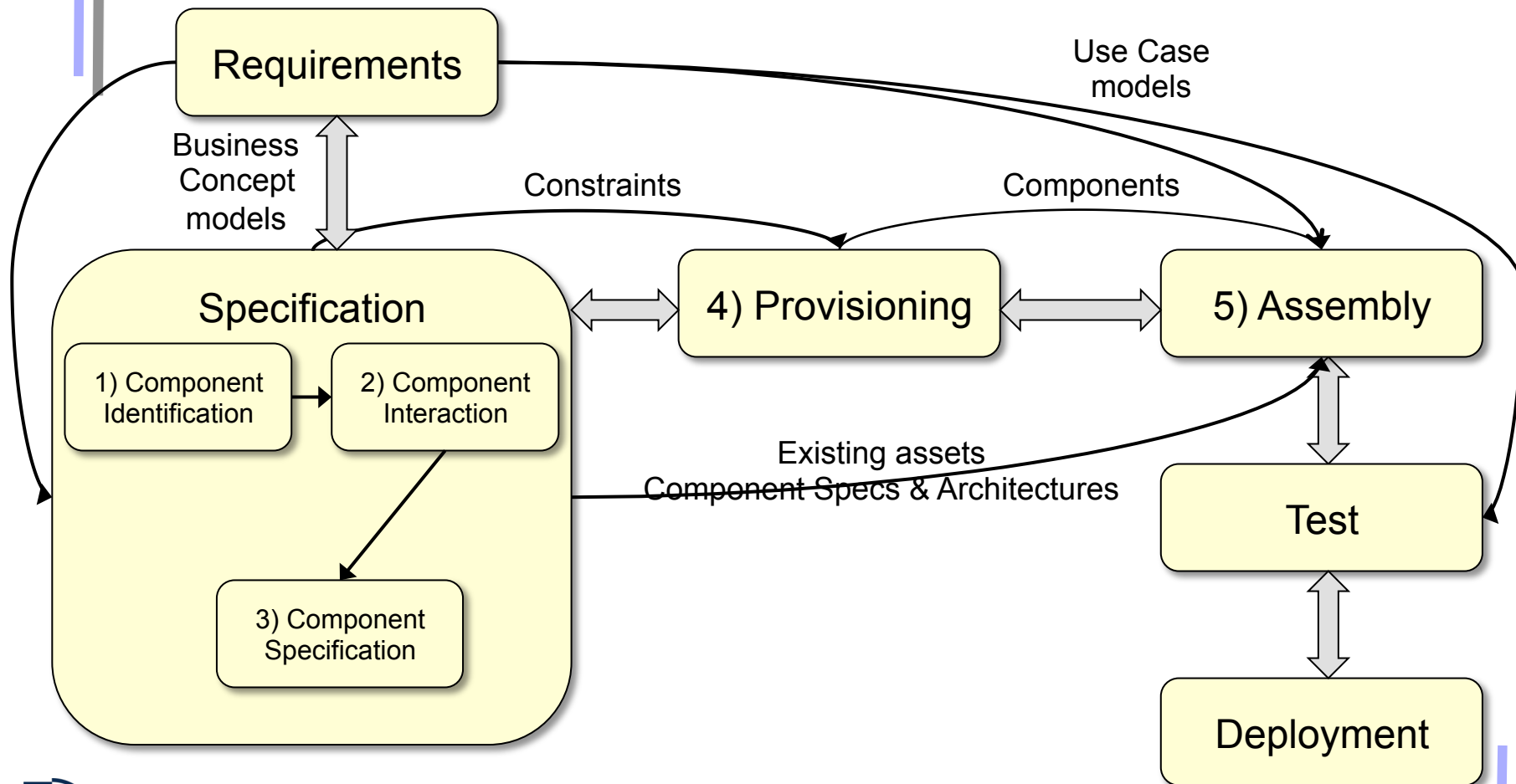
- ▶ In the Cheesman-Daniels component model, a **business component** consists of a set of business objects and other business components (part-of relation)
- ▶ The smallest component is a *business object*
 - groups several interfaces together.
 - has several provided interfaces
 - has several required interfaces
 - The business objects are the logical entities of an application
 - Their interfaces are re-grouped on system components for good information hiding and change-oriented design
 - Has a specification containing all interfaces and contracts
 - Has an implementation
 - UML-CD are used (UML profile with stereotypes)

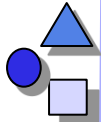
10.2. Identifying Business Components



Identifying Business Components with the Cheesman-Daniels CBSE Process

Overall development process

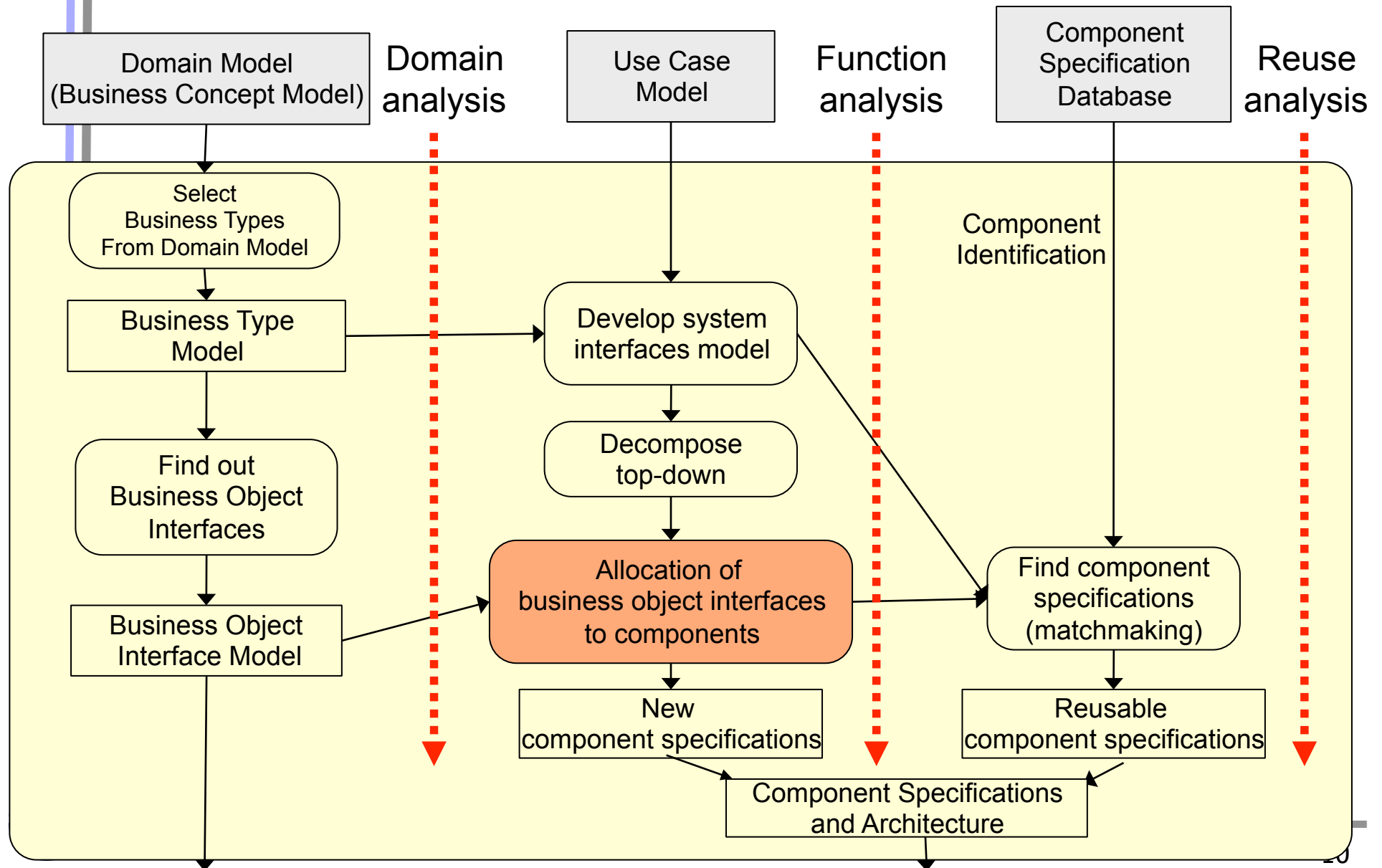




Artifacts of the Cheesman/Daniels Process

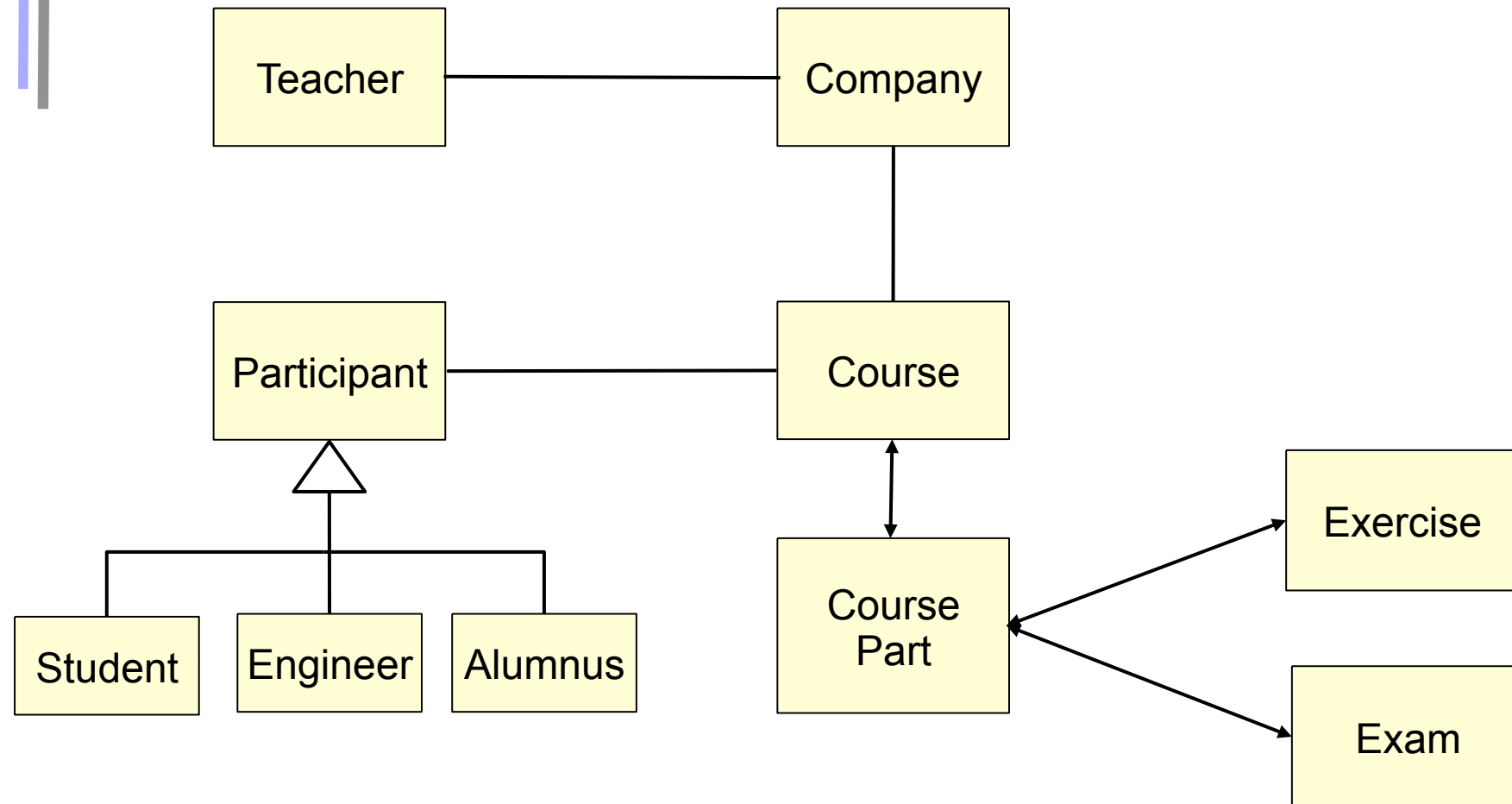
- ▶ Requirement artifacts:
 - *Business concept model (business model, domain model)*: describes the business domain (application domain)
 - *Use case model* (requirements model)
- ▶ System artifacts, derived from the business concept model:
 - *Business type model*, derived from domain model.
 - Represents the system's perspective on the outer world (more attributes, refined class structures from the system's perspective)
 - *Business object interface model*, containing the business objects and all their interfaces
 - *Business object model*, derived from the business object interface model by adding operations
- ▶ System component artifacts
 - Component interface specifications: one contract with the client
 - Component interface information model (state-based model)
 - Component specifications: all interface specifications of a component plus constraints.
 - Component architecture: wiring (topology) of a component net.

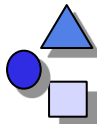
10.2.1 Component Identification (Step 1)



Ex.: Domain Model of a Course-Management System

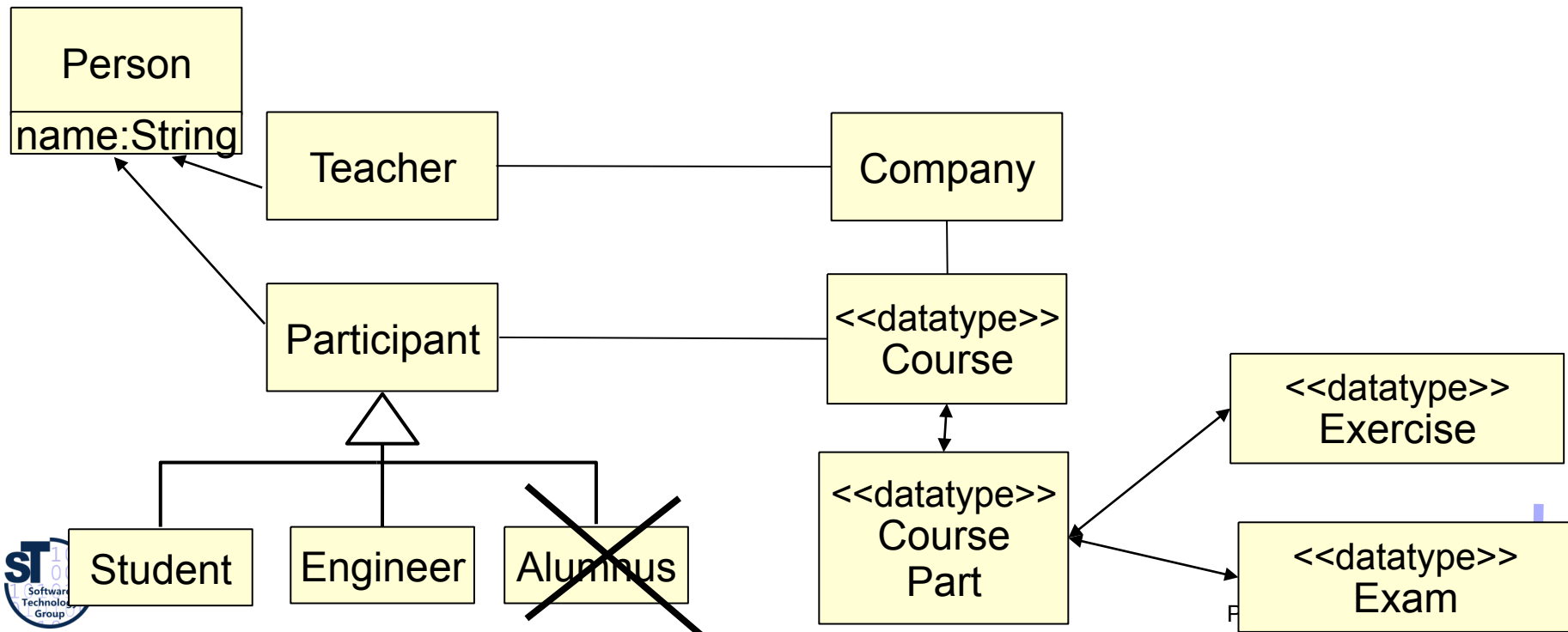
- ▶ Collects all concepts of the domain (aka business concept model)

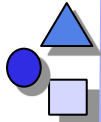




Business Type Model

- ▶ Defines system types from the domain model
 - Eliminates superfluous concepts
 - Adds more details
 - Distinguish datatypes (passive objects)



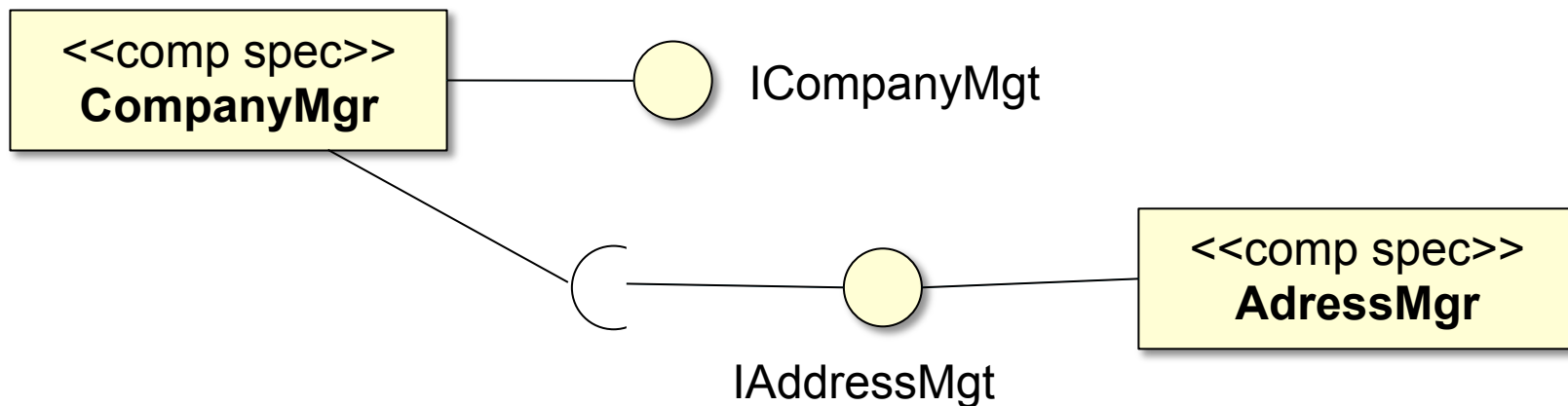


Component Specification with UML Components

- A UML component has *provided* and *required* interfaces
 - Provided interfaces are using „Lollipop“ notation
 - Required interfaces use „plug“ notation



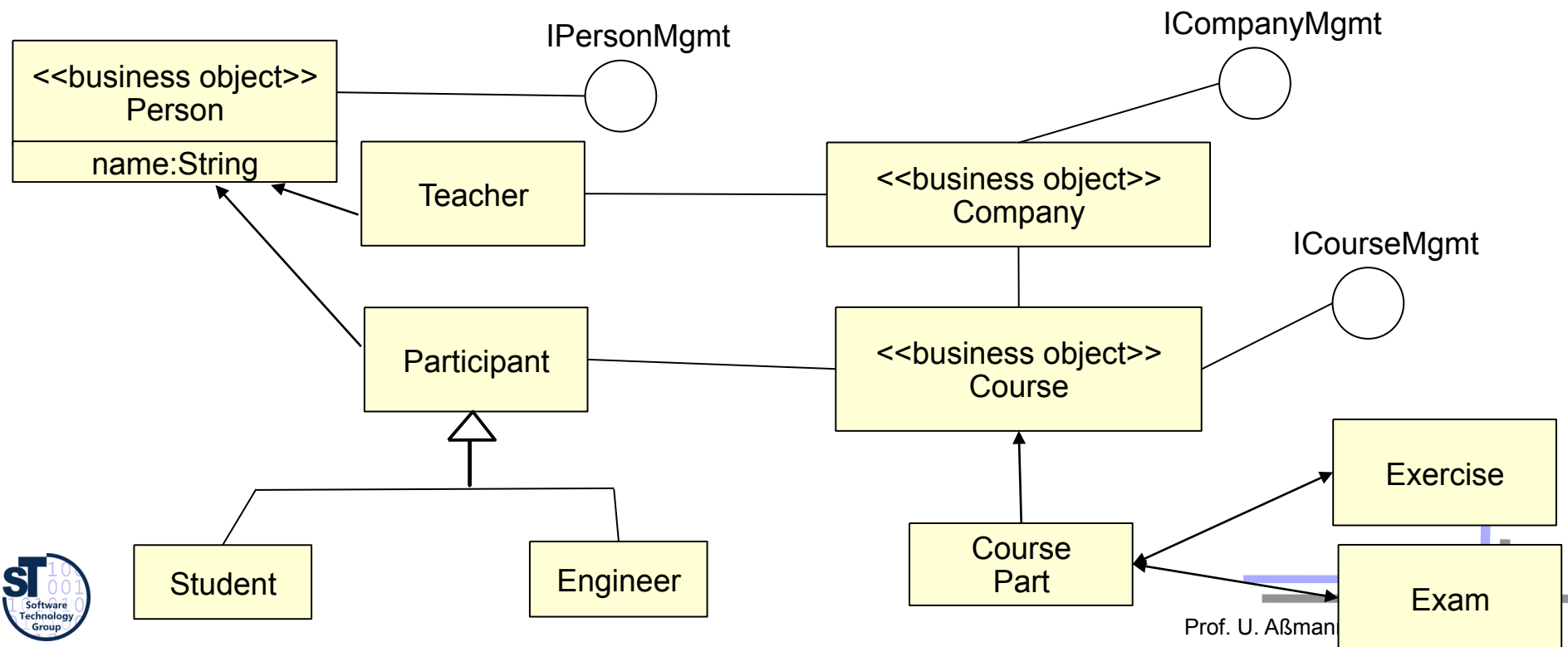
- Some components are required to use specific other interfaces

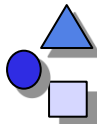




Business Object Interface Model

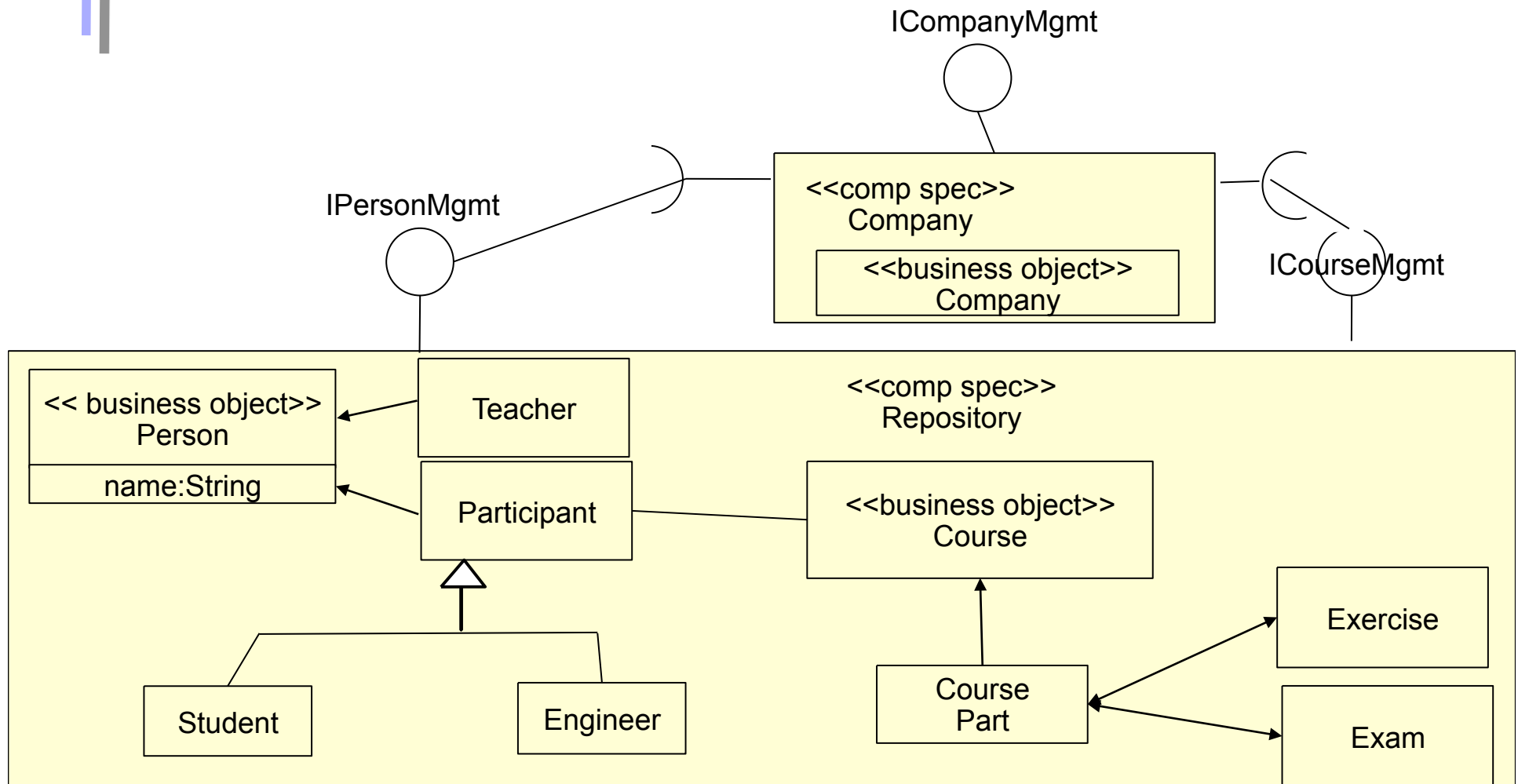
- ▶ Identifies business objects from the business type model
 - And defines *management interfaces* for them
 - Here, only Company, Course, Person are business objects, all others are dependent types

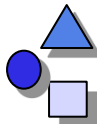




Component Identification (Version 0.1)

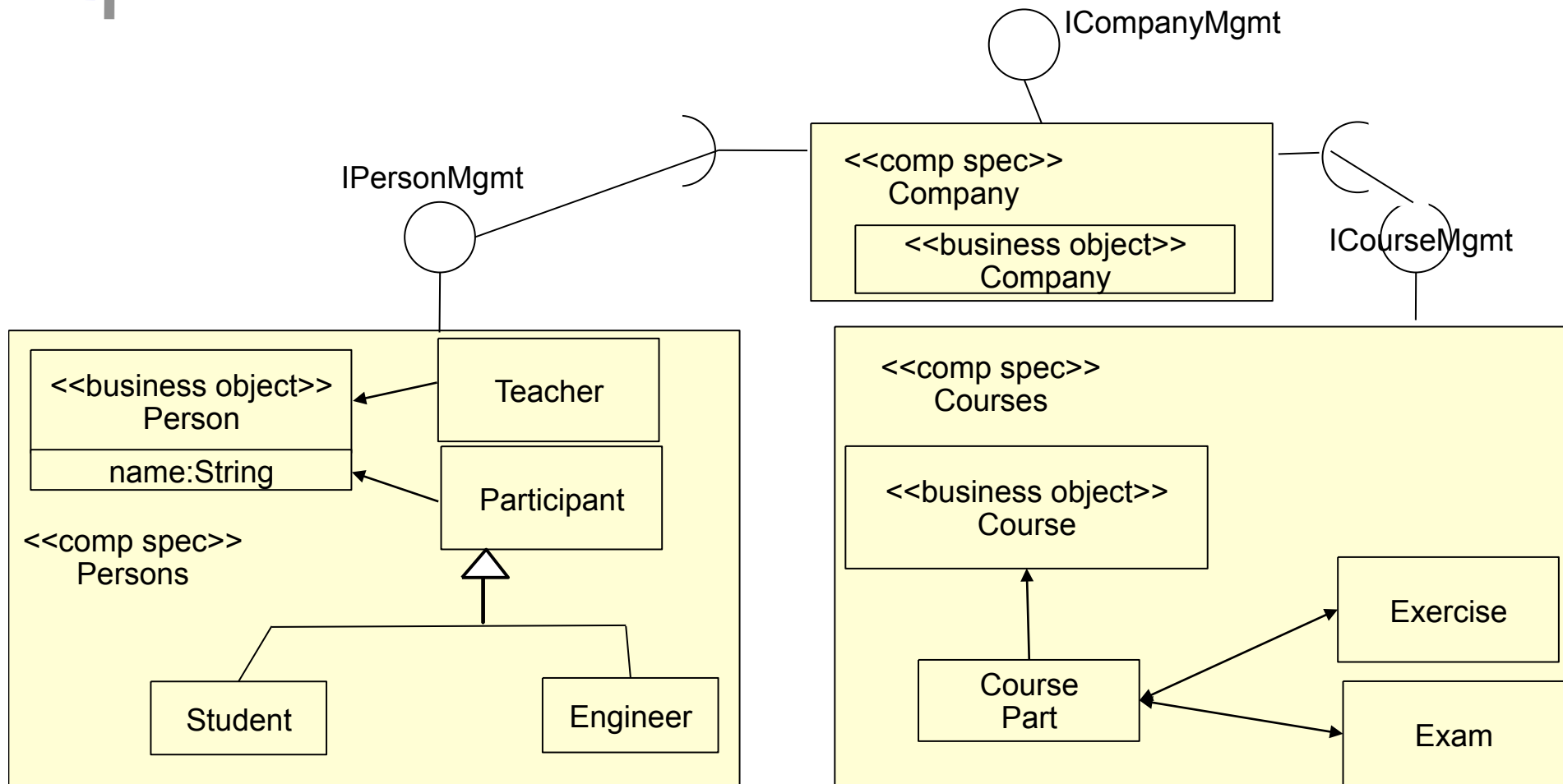
- ▶ Group classes and interfaces into reusable components





Alternative Component Identification (0.1)

- ▶ Group classes and interfaces into components
- ▶ Person management might be reusable

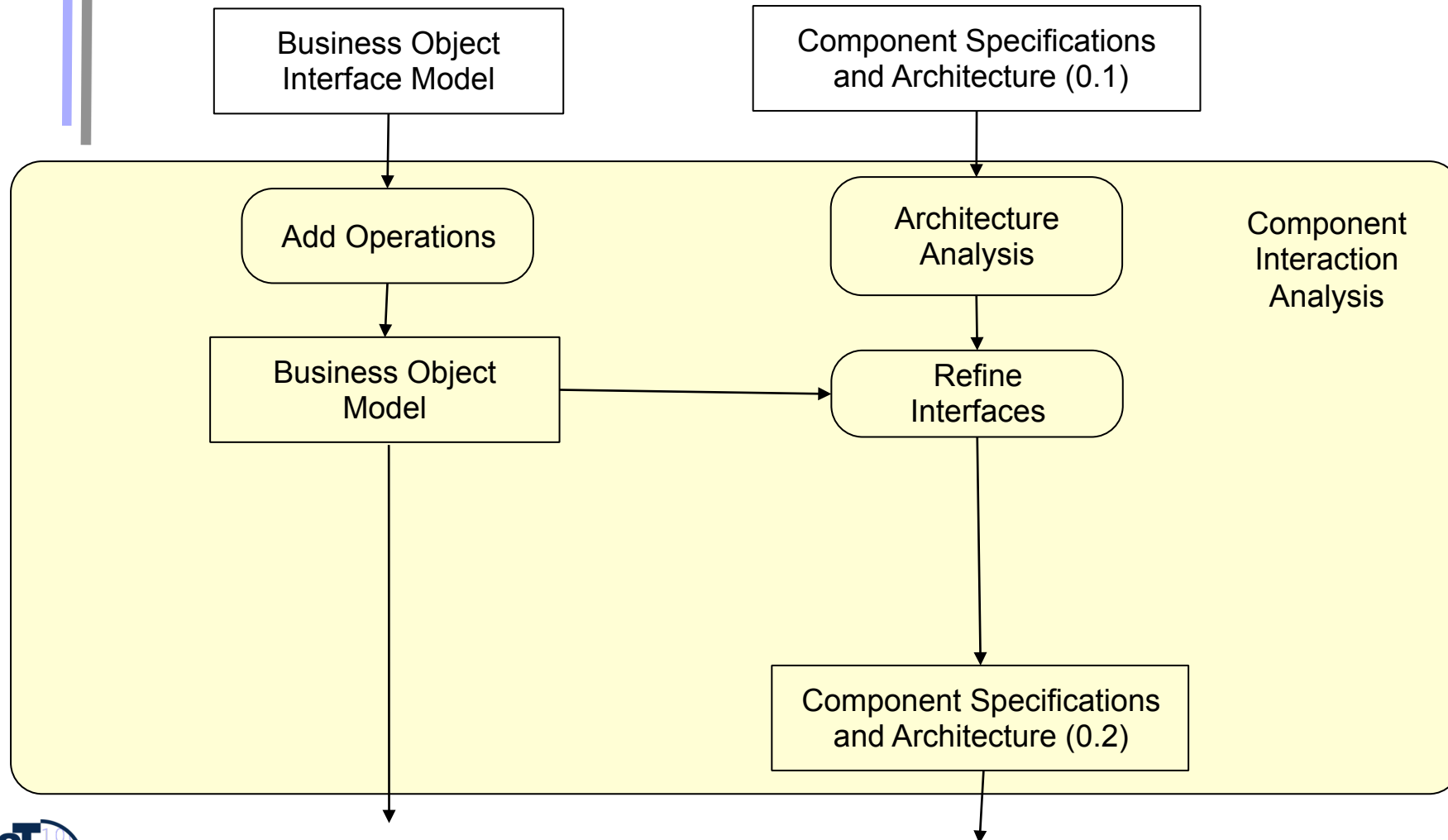


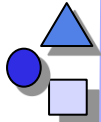


Component Identification

- ▶ The **component identification** subprocess attempts to
 - Create a business object interface model from the domain model (still without methods)
 - Attempts to group these interfaces to initial *system component specifications*
 - The grouping is done according to
 - *information hiding*: what should a component hide, so that it can easily be exchanged and the system can evolve?
 - *Reuse considerations*: which specifications of components are found in the component specification repository, so that they can be reused?
- ▶ There is a tension between business concepts, coming from the business domain (problem domain), and system components (solution domain). This gap should be bridged.

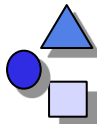
10.2.2 Component Interaction Analysis (Step 2)



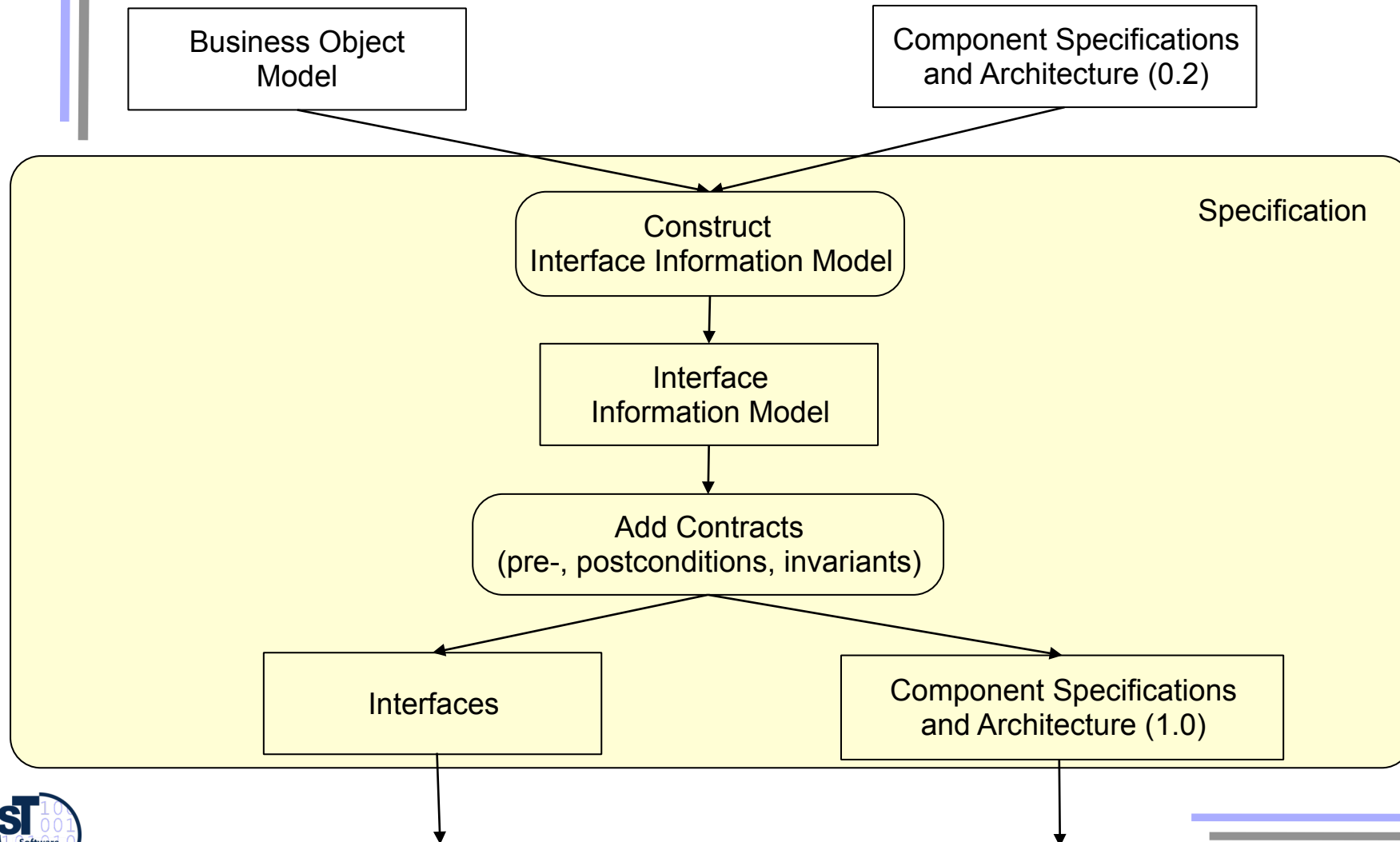


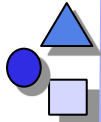
Component Interaction Analysis

- ▶ Is basically a refinement of the first stage
 - Removing,
 - Regrouping,
 - Augmenting,
 - Producing component specifications and wirings in a version 0.2
- ▶ Additionally, operations are added to business object interfaces
 - And mapped to internal types.



10.2.3 Component Specification (Step 3)

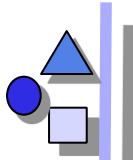




Component Specification (Step 3)

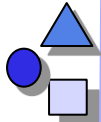
- ▶ Specification of declarative contracts for UML components in OCL
- ▶ Invariant construction:
 - Evaluate business domain rules and integrity constraints
 - Example:

```
context r: Course
-- a course can only be booked if it has been allocated in
the company
inv: r.bookable = r.allocation->notEmpty
```
- ▶ Pre/Postconditions for operations
 - Can only be run on some state-based representation of the component
 - Hence, the component must be modeled in an *interface information model*
 - Or: be translated to implementation code (e.g. Java using an OCL2Java Compiler)



10.2.4. Provisioning (Realization, Implementation) (Step 4)

- ▶ Provisioning selects component implementations for the specifications
 - Choosing a concrete implementation platform (EJB, CORBA, COM+, ...)
 - Look up component implementations in implementation repositories
 - Write adapters if they don't fit exactly
 - Program missing components
 - Store component implementations and specifications in database for future reuse



10.2.5 Assembly (Step 5)

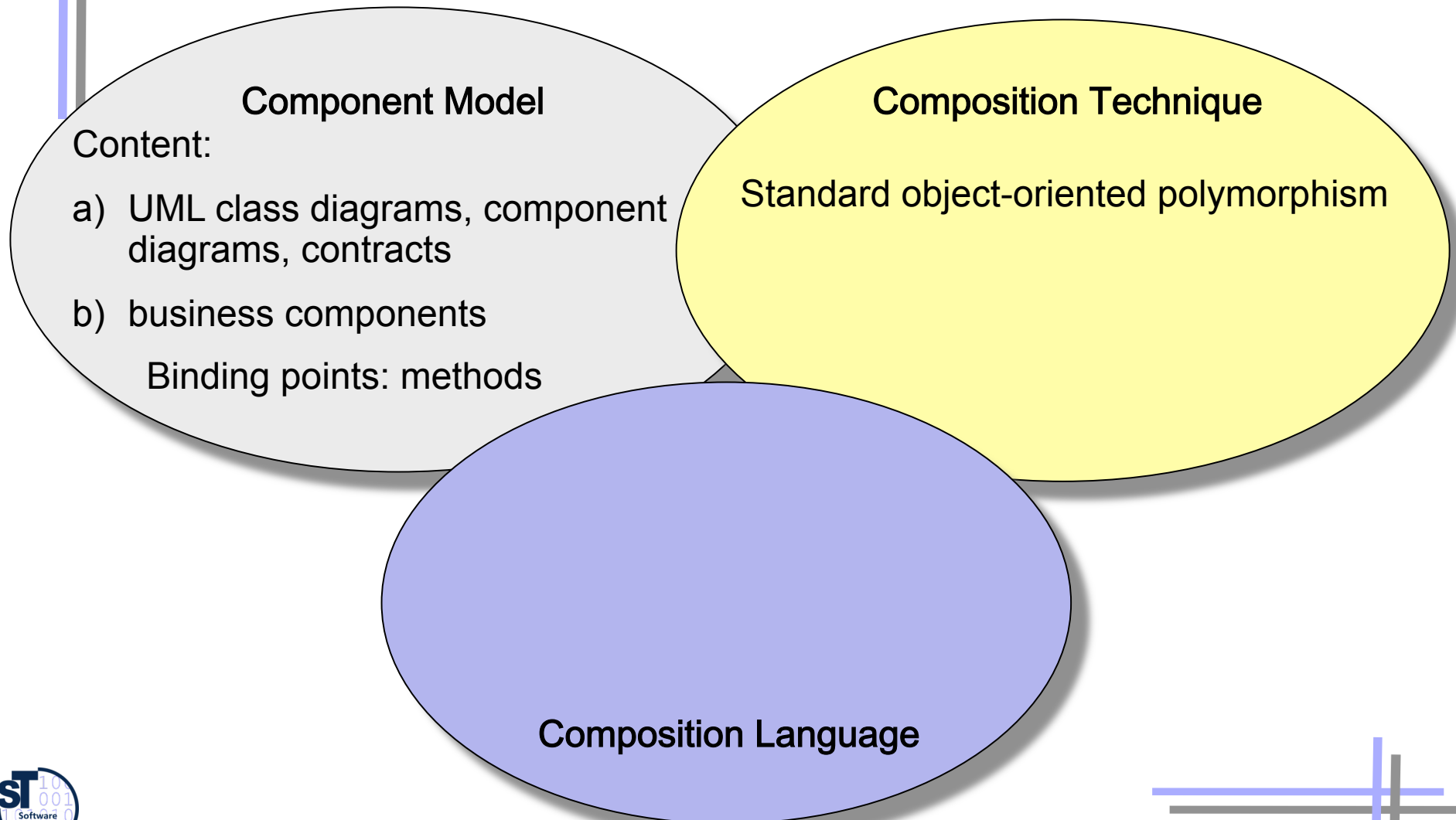
- ▶ Puts together architecture, component specifications and implementations, existing components
 - We will see more in the next lectures



Weaknesses

- ▶ No top-down decomposition of components
 - part-of relationship is not really supported
- ▶ Reuse of components is attempted, but
 - Finding components is not supported (see companion lecture)
 - Metadata
 - Facet-based classification

Cheesman-Daniels' Business Component Model as Composition System





The End