Software Engineering Essentials

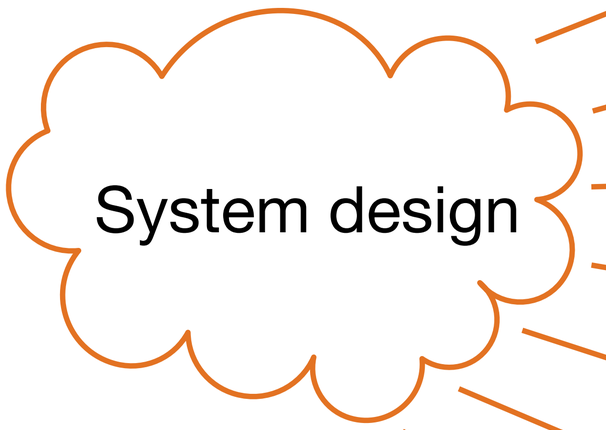
Summary 2

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

Learning Goals

1) Review the main concepts of the course
2) Understand the transition from a problem statement to a software release
3) Remember the key facts about applied software engineering

System design



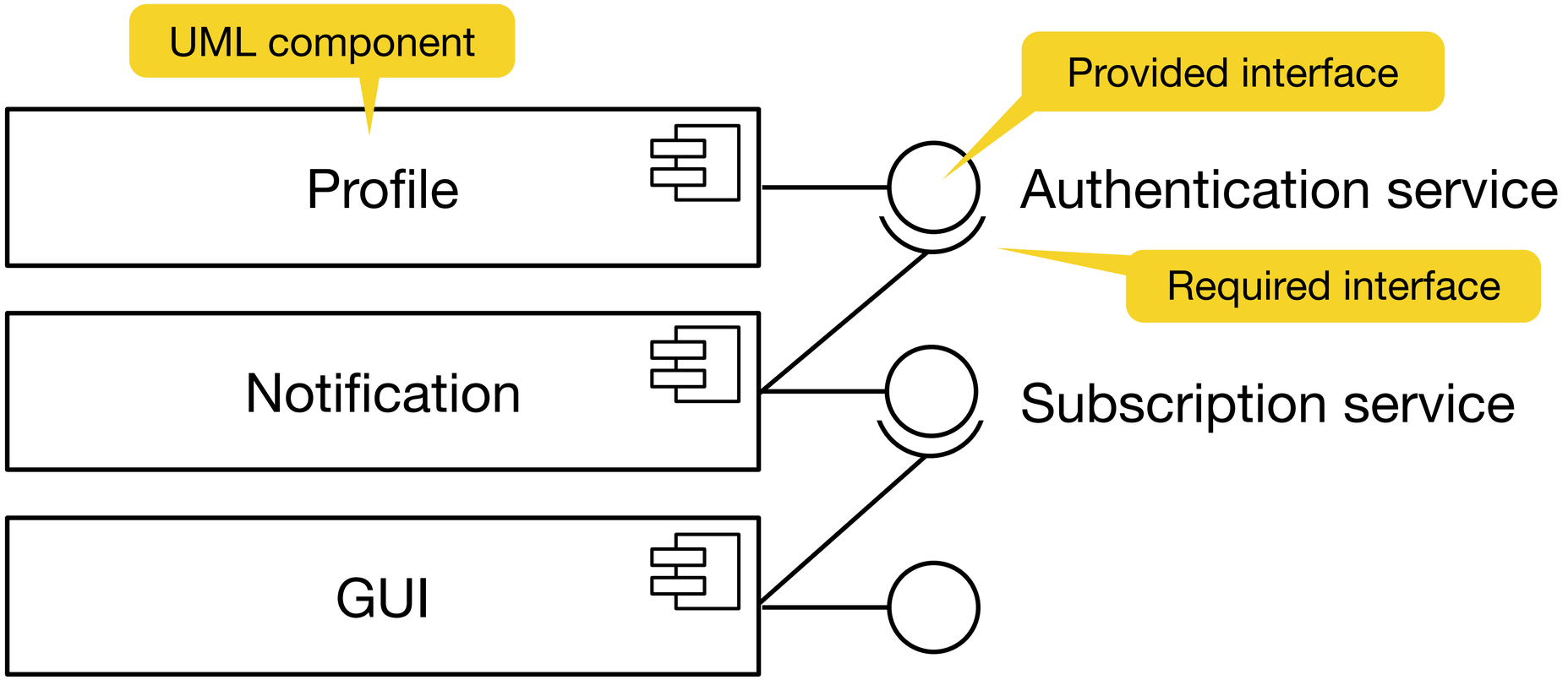
1. Design goals
2. Subsystem decomposition
3. Identify concurrency
4. Hardware / software mapping
5. Persistent data management
6. Global resource handling
7. Software control
8. Boundary conditions

Derived from non-functional requirements

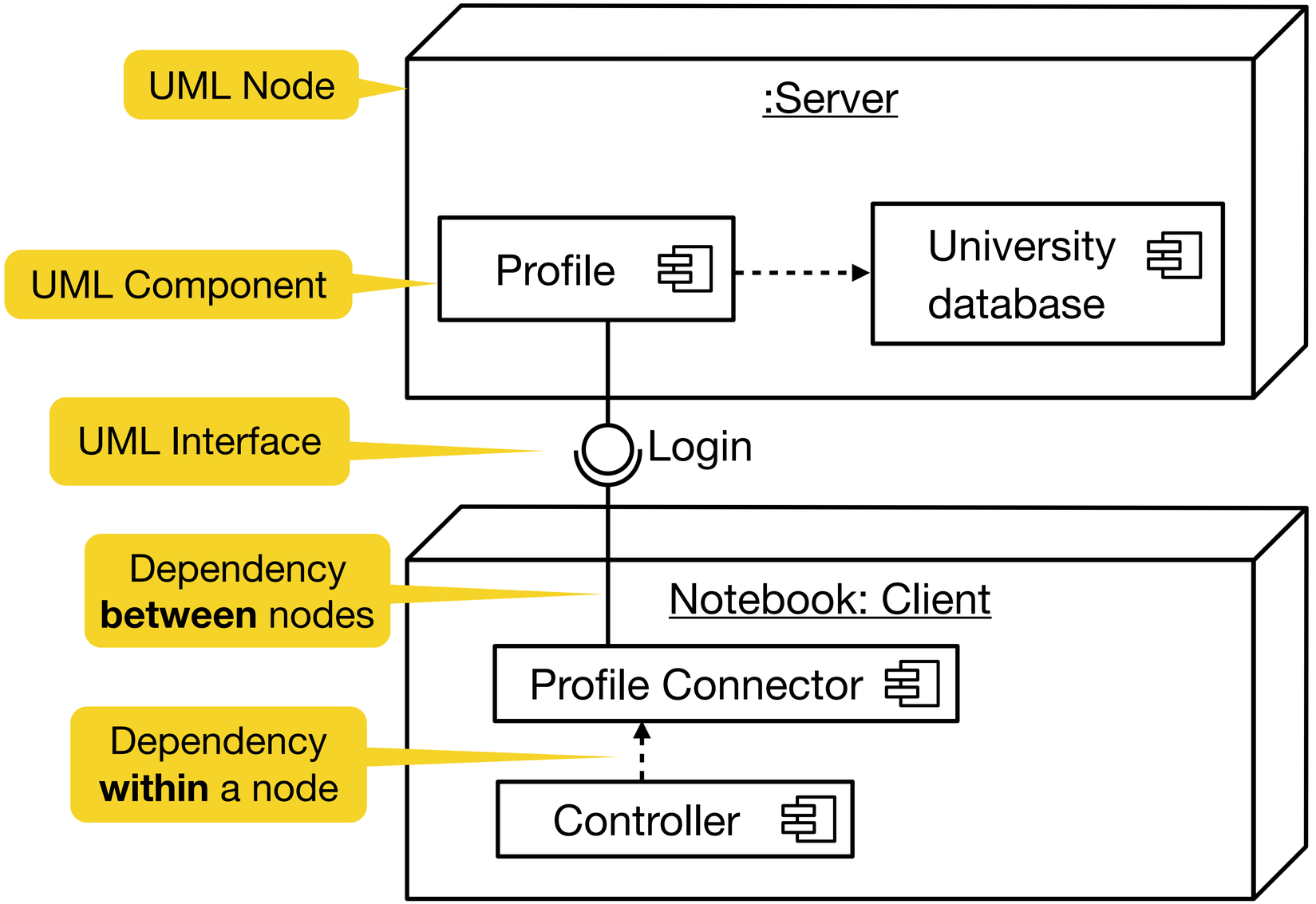
Derived from the functional model (UML component diagram)

Derived from the object model (UML deployment diagram)

UML component diagram

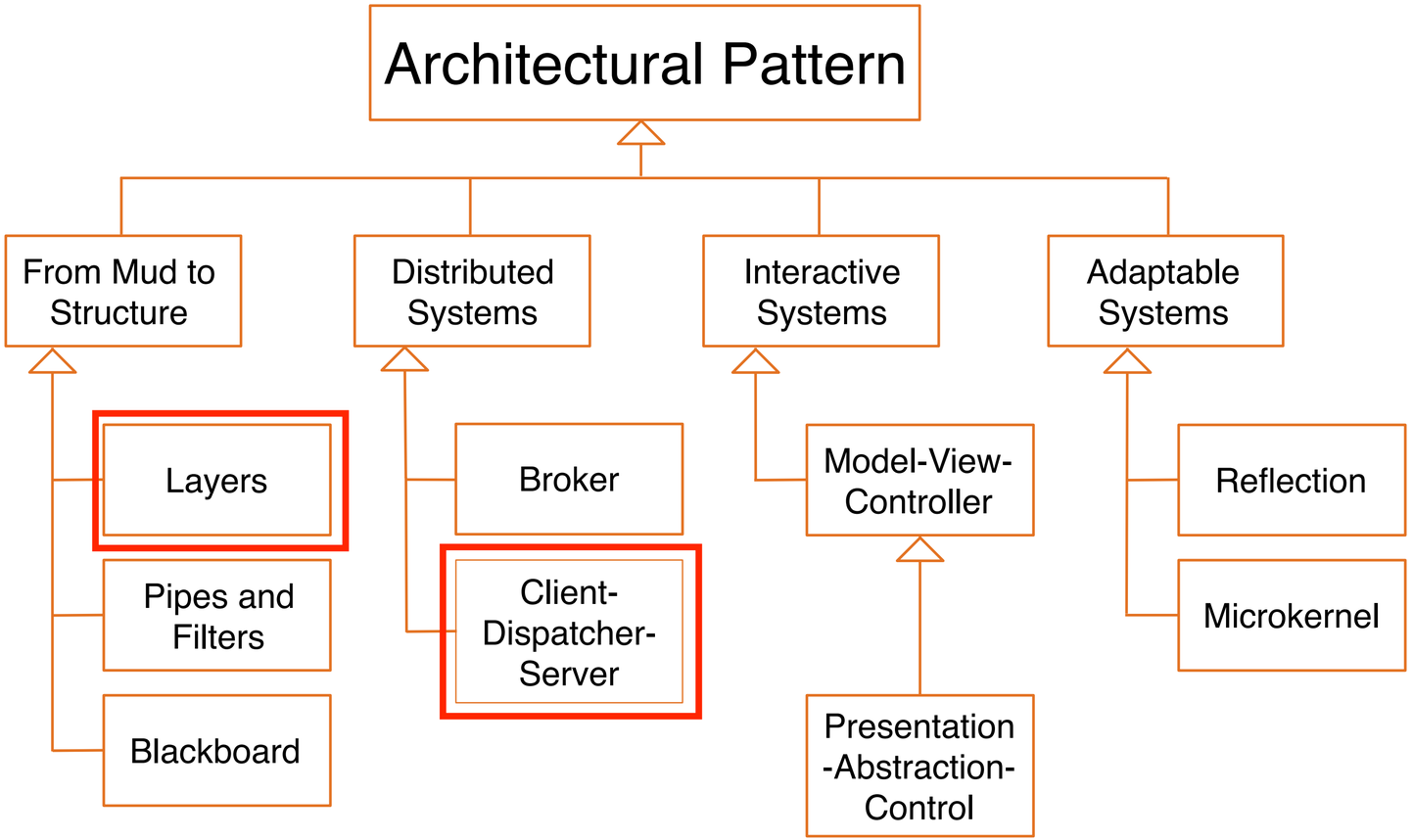


UML deployment diagram



Architectural patterns

- Architectural Style = Architectural Pattern: A pattern for a subsystem decomposition
- Software Architecture: Instance of an architectural style / pattern



Object design

Purpose:

- Prepare for the implementation of the system model based on design decisions
- Identify reuse possibilities (buy vs build)

4 main activities

1) Reuse: Identification of existing solutions
   - Use of inheritance
   - Use of design patterns
2) Interface specifications
   - Describe precisely each class interface

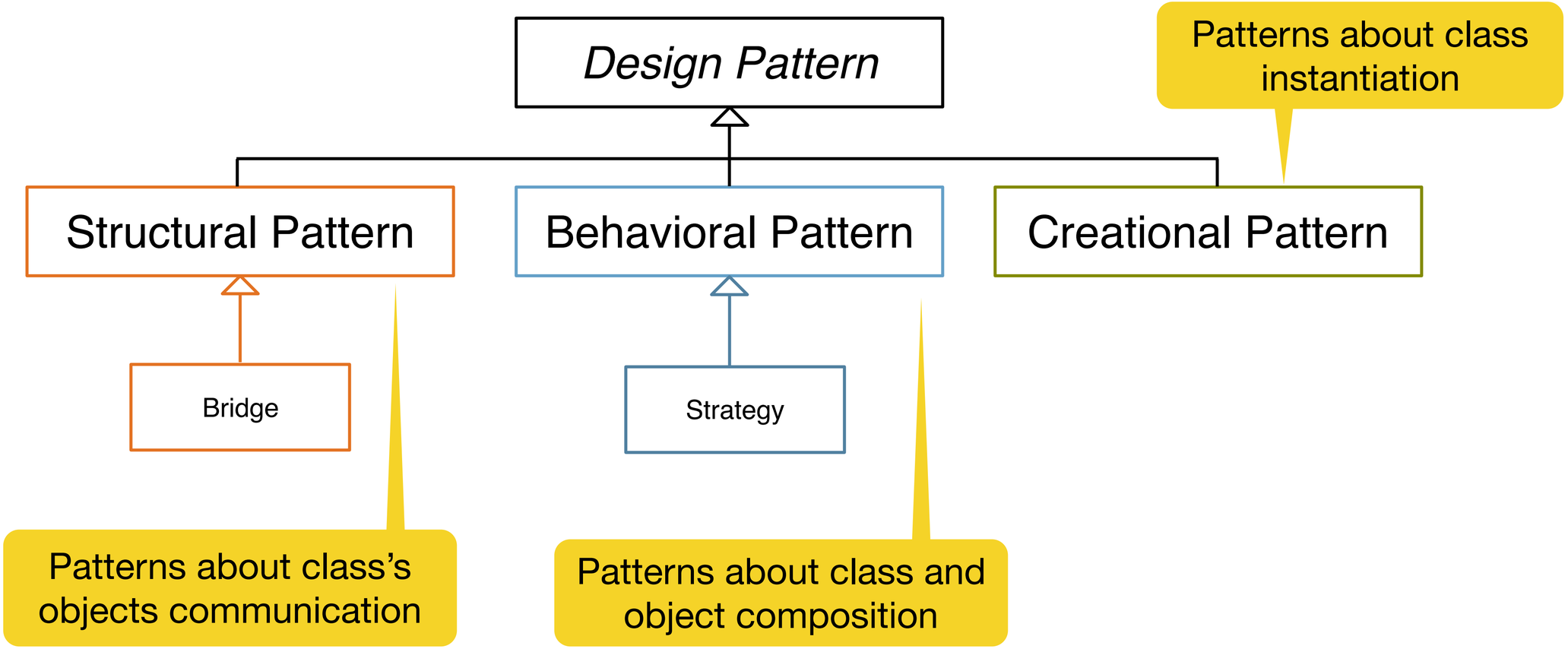
Focus on reuse and specification

3) Object model restructuring
4) Object model optimization

Towards mapping models to code

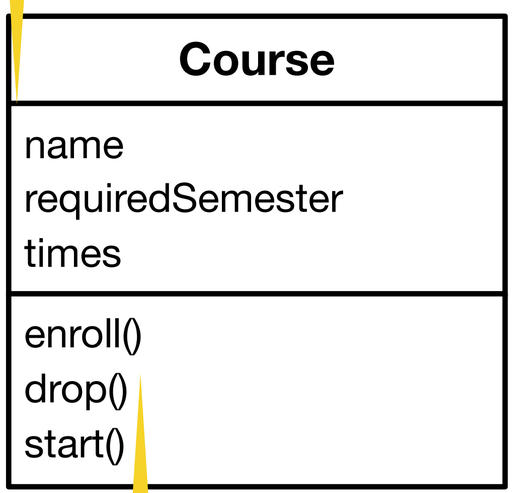
Design patterns

- Generalize of detailed design knowledge from existing systems
- Provide a shared vocabulary and examples of reusable design



UML class diagram (object design)

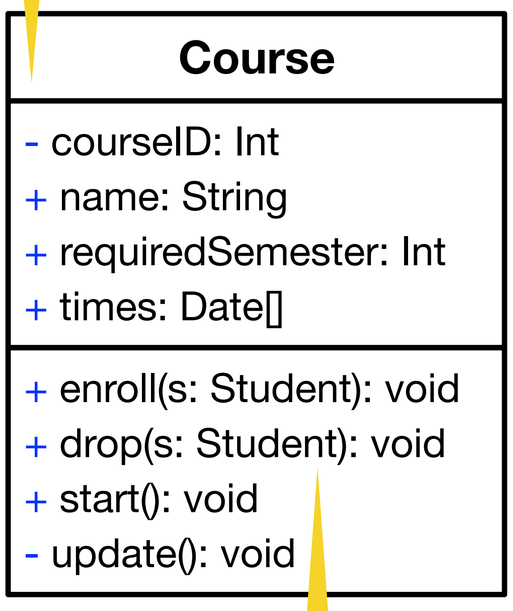
During **analysis**: attributes and methods **without** visibility information



During **analysis**: methods **without** signature

Analysis (application domain language)

During **object design**: we specify the visibility for each attribute and method



During **object design**: we specify the signature of each method

Object design (solution domain language)

```
public class Course {

  private Integer courseId;
  public String name;
  public Integer requiredSemester;
  public Date[] times;


  public void enroll(Student s) {…}
  public void drop(Student s) {…}
  public void start() {…}
  private void update() {…}

}
```

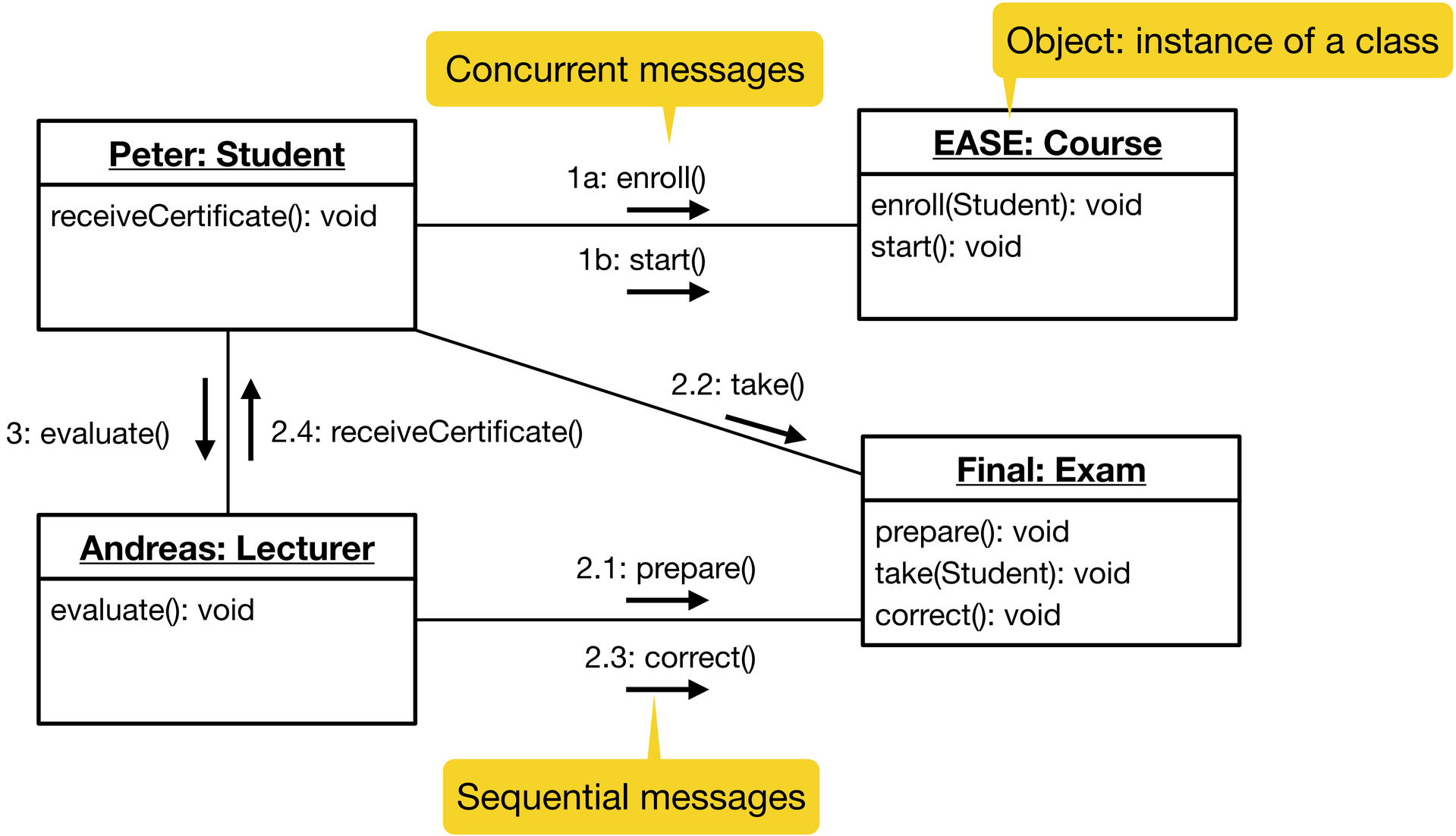
type

signature

visibility

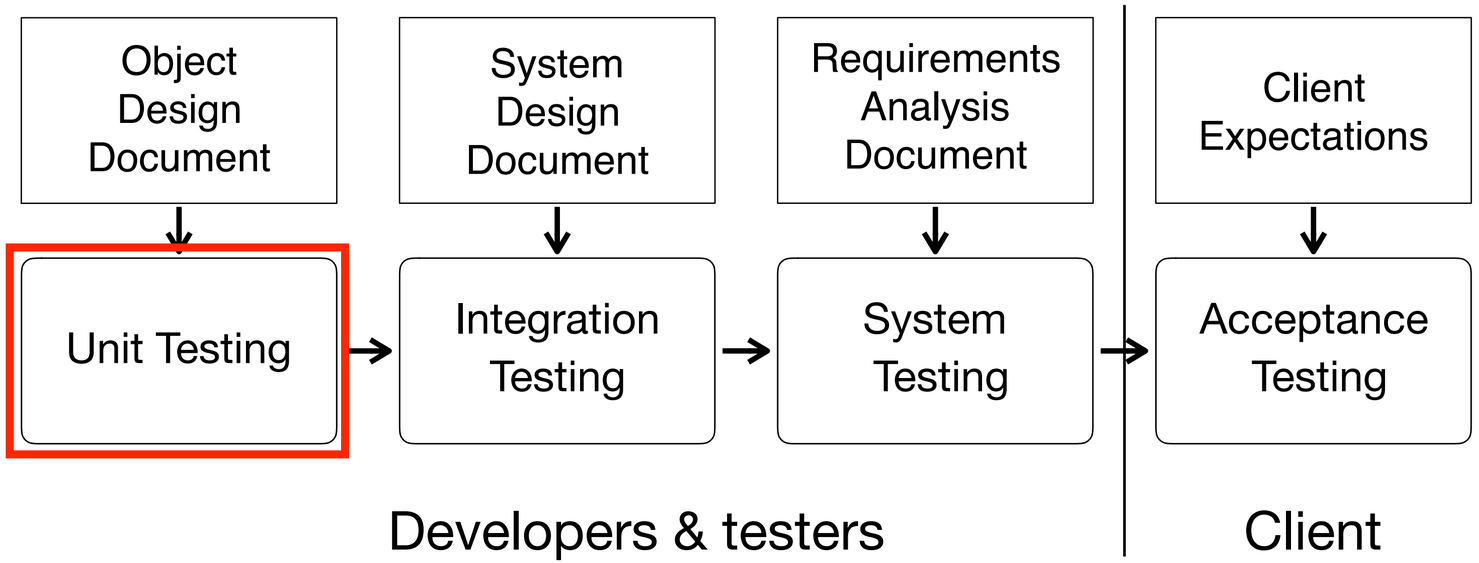
Implementation

UML communication diagram



Testing

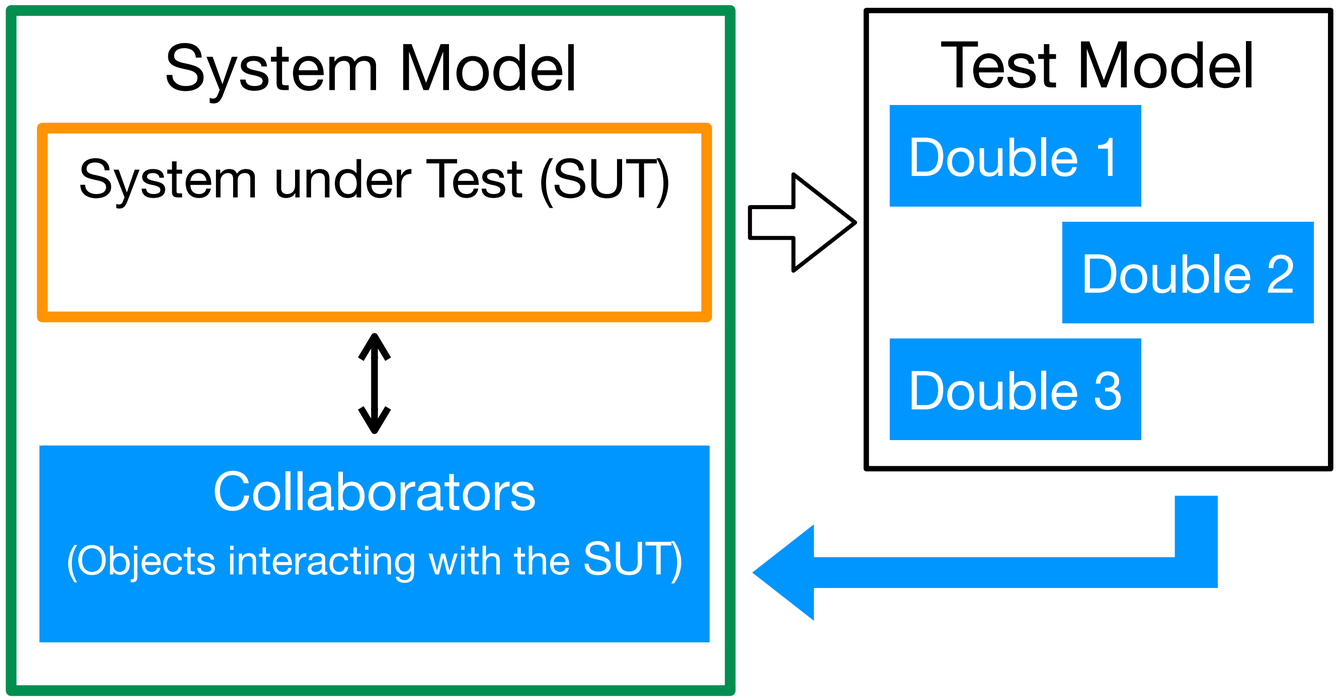
- **Failure:** any deviation of the observed behavior from the specified behavior (also often called “crash”)
- **Error:** the system is in a state such that further processing by the system can lead to a failure (often called “erroneous state”)
- **Fault:** the mechanical or algorithmic cause of an a error (often called “bug”)



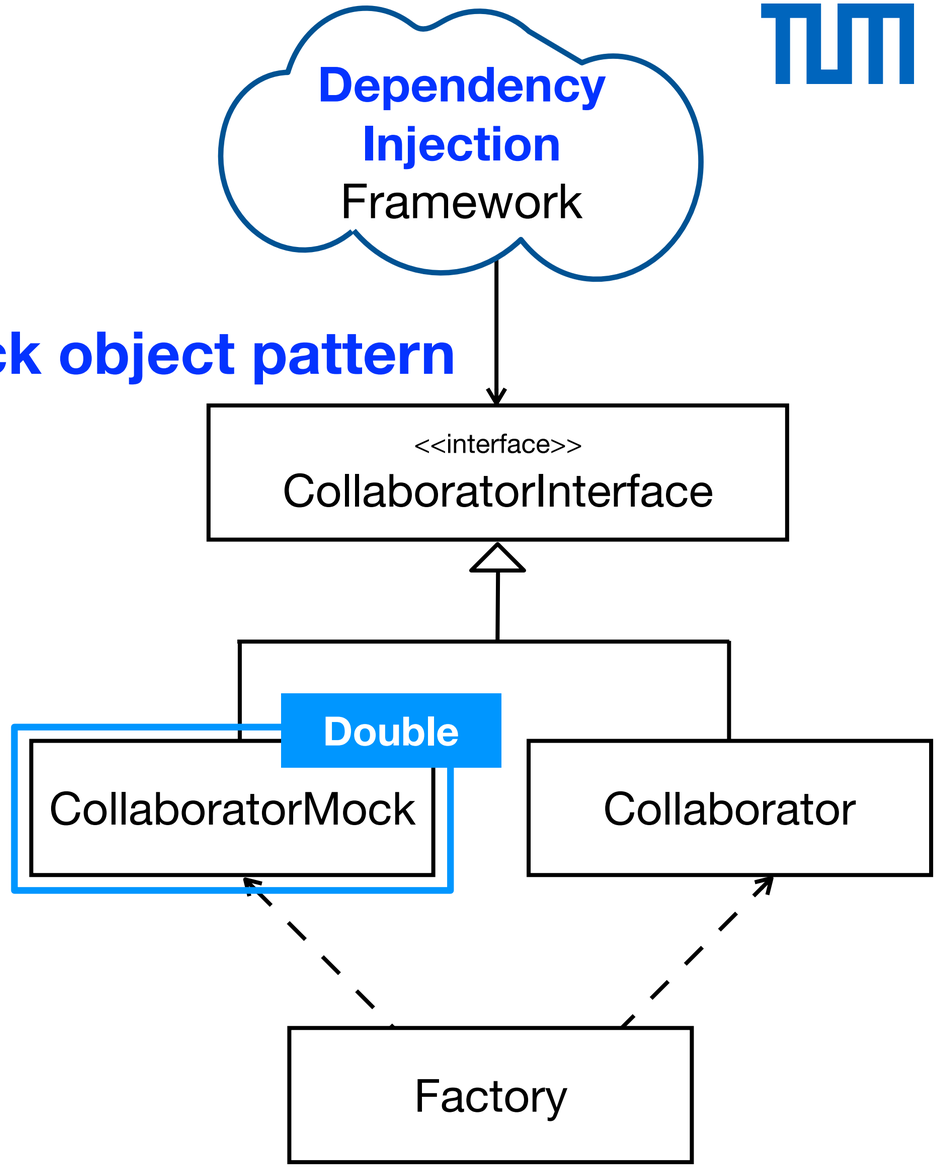
Regression Testing: Testing after each change

Testing patterns

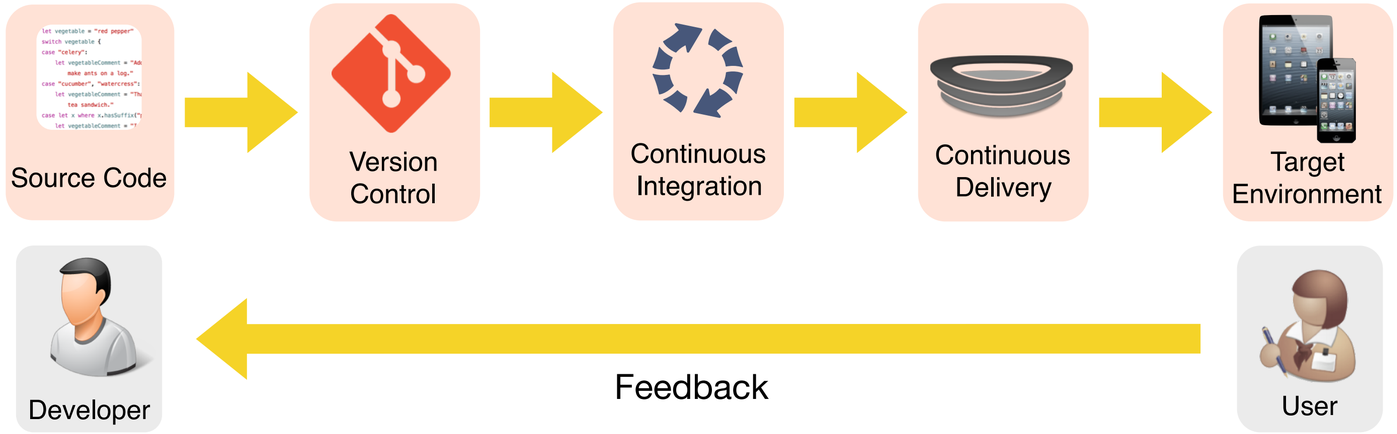
Model based testing



Mock object pattern



Build and release management



- Continuous integration: integrate frequently (at least daily)
- Continuous delivery: produce valuable software in short cycles and ensure that the software can be reliably released at any time.
- Continuous deployment: every change is deployed automatically

Software Engineering Essentials

Summary 2

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