Chapter 1: Introduction

Object-**O**riented
Software **C**onstruction

Prof. Dr. Armin B. Cremers
Tobias Rho & Daniel Speicher & Holger Mügge

The team
... for further questions

Official Lecturer:

- Prof. Dr. Armin B. Cremers (abc@cs.uni-bonn.de)

Lecture Organization:

- Tobias Rho(rho@cs.uni-bonn.de)

- Daniel Speicher (dsp@cs.uni-bonn.de)

- Holger Mügge (muegge@cs.uni-bonn.de)

Exercises:

- Mahmoud El-Gayar (elgayyar@iai.uni-bonn.de)
- Matthias Berg (bergm@iai.uni-bonn.de)

Objectives of the Class

- Introduction to
  - Technical aspects of building complex software systems
  - Object-Oriented Modeling with the Unified Modeling Language
  - The Complete Software Lifecycle
  - Configuration & Rationale Management
- Advanced Topics are covered in a follow-up lecture
  - Advanced Topics of Software Construction
  - Deepens Requirements Engineering, e.g. Requirements Writing
  - Software Processes
  - Software Architectures
  - Advanced Technologies
    - ➔ Aspect-oriented Software Development (Separation of Crosscutting Concerns)
    - ➔ Model-Driven Architecture

Acquire Technical Knowledge

- Learn standard modeling language
  - UML 2.0 (Unified Modeling Language)
- Learn standard modeling methods
- Learn how to use tools
  - Eclipse
  - Subversion
  - CASE (Computer Aided Software Engineering)
- Improve your knowledge in Java (6.0)
- Learn how to use
  - Design Patterns
  - Refactoring

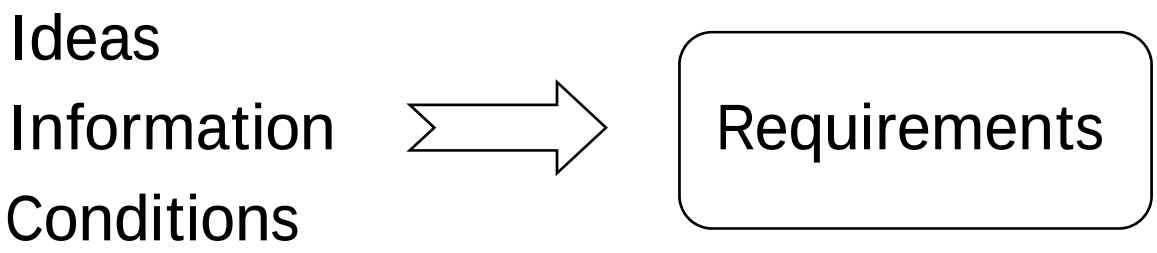
Readings

- Required:
  - Bernd Bruegge, Allen Dutoit: “Object-Oriented Software Engineering: Using UML, Patterns, and Java”, Prentice Hall, 2003.
- Recommended:
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: “Design Patterns”, Addison-Wesley, 1996.
  - Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide, V.2.0”, Addison Wesley, 2005.
- Additional books may be recommended during individual lectures
- Some sources will be made available on the web site

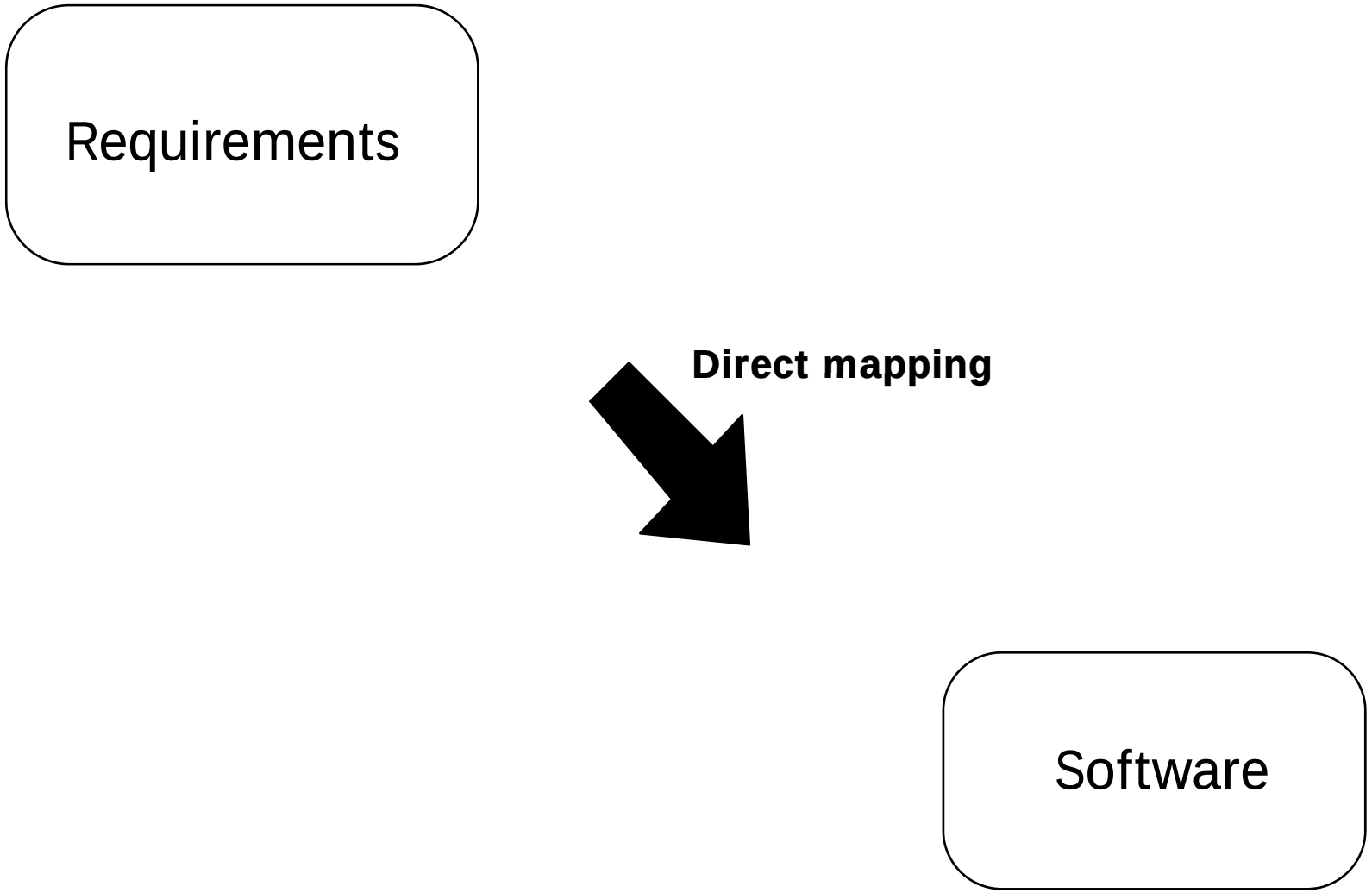
Outline of Today's Lecture

- Perspectives on Software Engineering
- Modeling complex systems
  - Functional vs. object-oriented decomposition
- Software lifecycle
- Overview of the class
- Concluding remarks

How to start with a software project?



Mapping requirements to Software
A good practice?



Perspectives on Software Engineering: Quality of Software

- “Bad software engineering” leads to functional misbehavior
- Updates are needed to fit the initial requirements
  - Increased costs
  - Delayed deployment
  - Unsatisfied customers

- Some examples:
  - Toll Collect (Germany): Technical Problems caused a delayed deployment of more than 2 years
  - Hartz IV-Software “A2II”: Regular Updates needed, high costs; Computation of the unemployment benefit cannot be guaranteed (http://www.heise.de/newsticker/meldung/78233)

→ The average software product released on the market is not error free

Perspectives on Software Engineering: Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

- **complex** and **huge** software systems
- with a given budget
- before a given deadline

while change occurs.

Perspectives on Software Engineering:
What makes up a software engineer?

- Computer Scientist (Researcher)
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite amount of time... (in general *no project*)
- Programmer
  - Mainly involved in the technical realization of software
- **Software Engineer**
  - Has to work in and understand multiple application domains
  - Must have technical and managerial background
  - Covers many (all) phases in software lifecycle in a *project*

Perspectives on Software Engineering:
A Problem Solving Activity

General Procedure for Problem-Solving activity:

- Formulate the problem
- Analyze the nature of problem and break the problem into pieces
- Search for solutions/Identify the most appropriate solutions
- Specify the solutions
- Aggregate the solutions

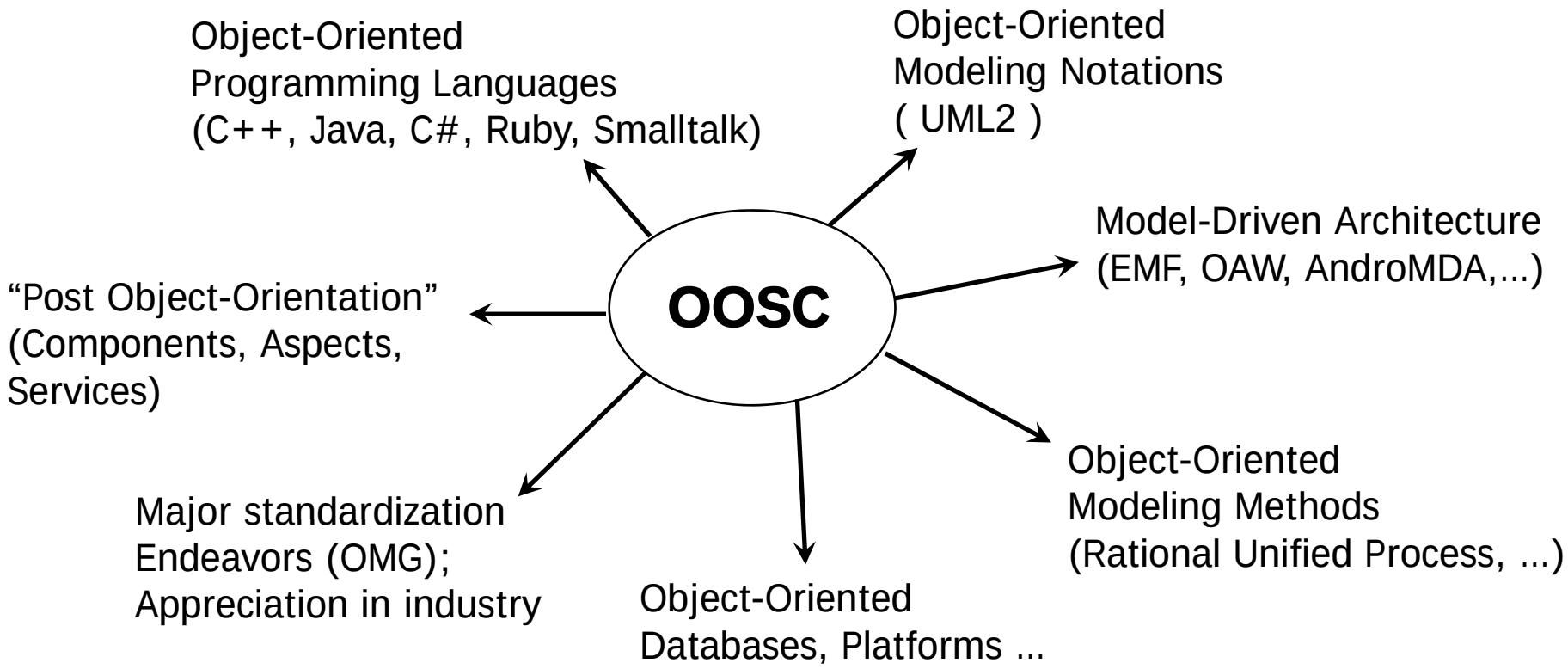
Perspectives on Software Engineering:
A Problem Solving Activity

Problem solving needs:

- Notation
  - Graphical or textual set of rules for representing a model
- Methods:
  - Repeatable technique that specifies the steps for solving a specific problem
- Methodologies:
  - Collection of methods for solving a class of problems. Specifies how and when each method should be used
- Tools:
  - Instrument or automated systems to accomplish a method
- Knowledge Acquisition
  - Nonlinear process: addition of new knowledge may invalidate old knowledge
- Rationale Management
  - Capturing the context in which decisions were made and the rationale behind these decisions

Object-Oriented Software Construction
Overview

- Today's mainstream development methodology in Software Engineering. Influences:



Factors affecting the quality of a software system

- Complexity
  - Complex technologies (programming languages)
  - The development process is very difficult to manage
  - Domains are complex that no single person can understand it
  - Complex (unfeasible) requirements from clients
  - Fixing a bug causes another bug

- Change
  - Requirements need to be updated when errors are discovered and when developers have a better understanding of the application
  - Project constellation changes (staff turn-around)
  - Technological changes (new standards, languages)

Dealing with Complexity
Three Main methods

1. Abstraction (Modeling)
   → Abstract from complex systems and condition and build models
   → improves **understanding, reusability**

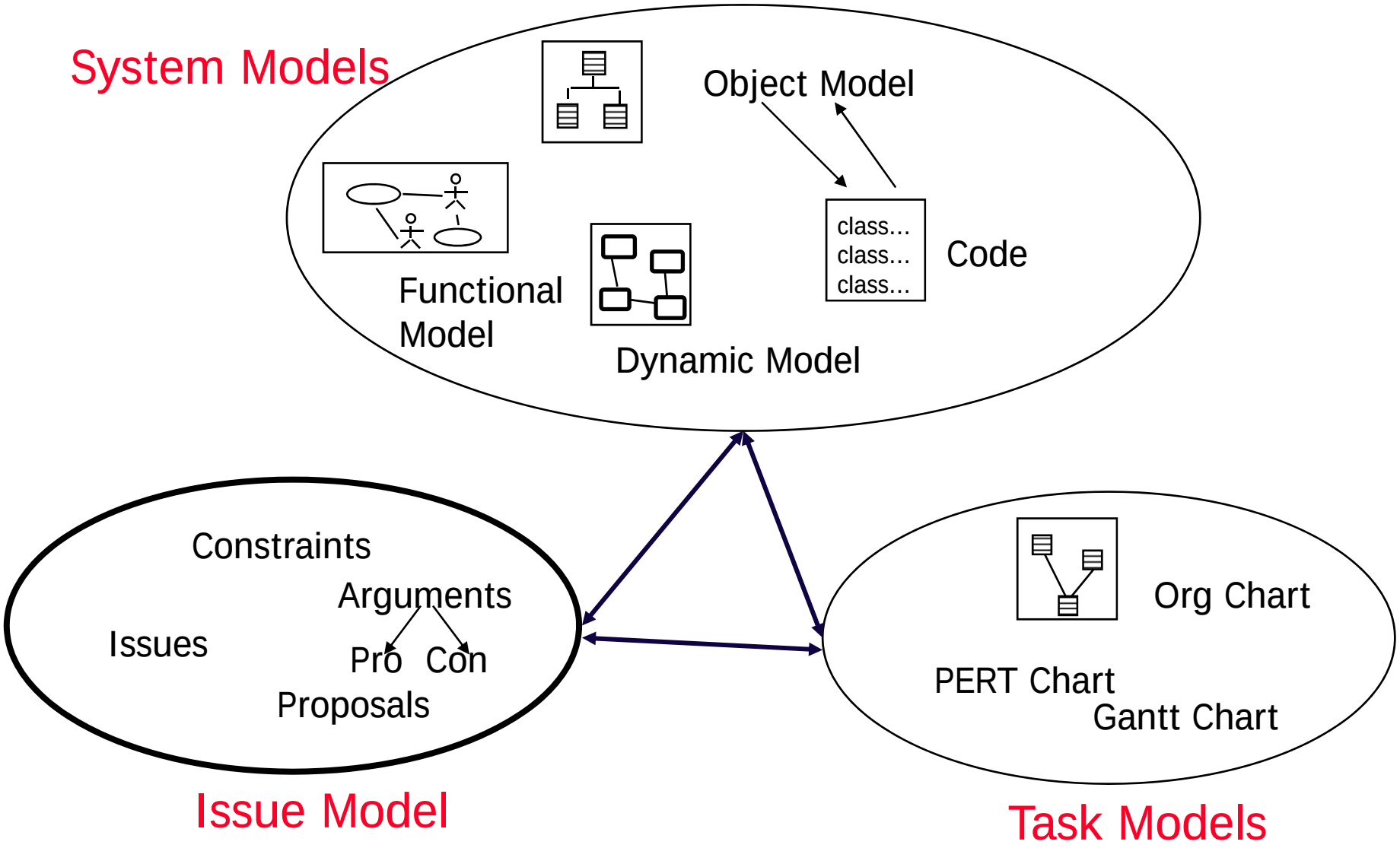
2. Decomposition
   → Divide your solution into independent pieces
   → improves **flexibility**, **effectiveness**

3. Hierarchy
   → Organize the system (and knowledge) in meaningful hierarchies
   → improves **understanding**

Abstraction:
Models are used to provide abstractions

- System Model:
  - Object Model: What is the **structure** of the system? What are the objects and how are they related?
  - Functional model: What are the **functions** of the system? How is data flowing through the system?
  - Dynamic model: How does the system react to **external events**? How is the event flow in the system ?
- Task Model:
  - PERT Chart: What are the **dependencies** between the tasks?
  - GANTT Chart: How can this be done within the **time limit**?
  - Org Chart: What are the **roles** in the project or organization?
- Issues Model:
  - What are the open and closed **issues**? What constraints were posed by the client? What resolutions were made?

Abstraction:
The “Triangle” of Modeling



Decomposition:
Overview

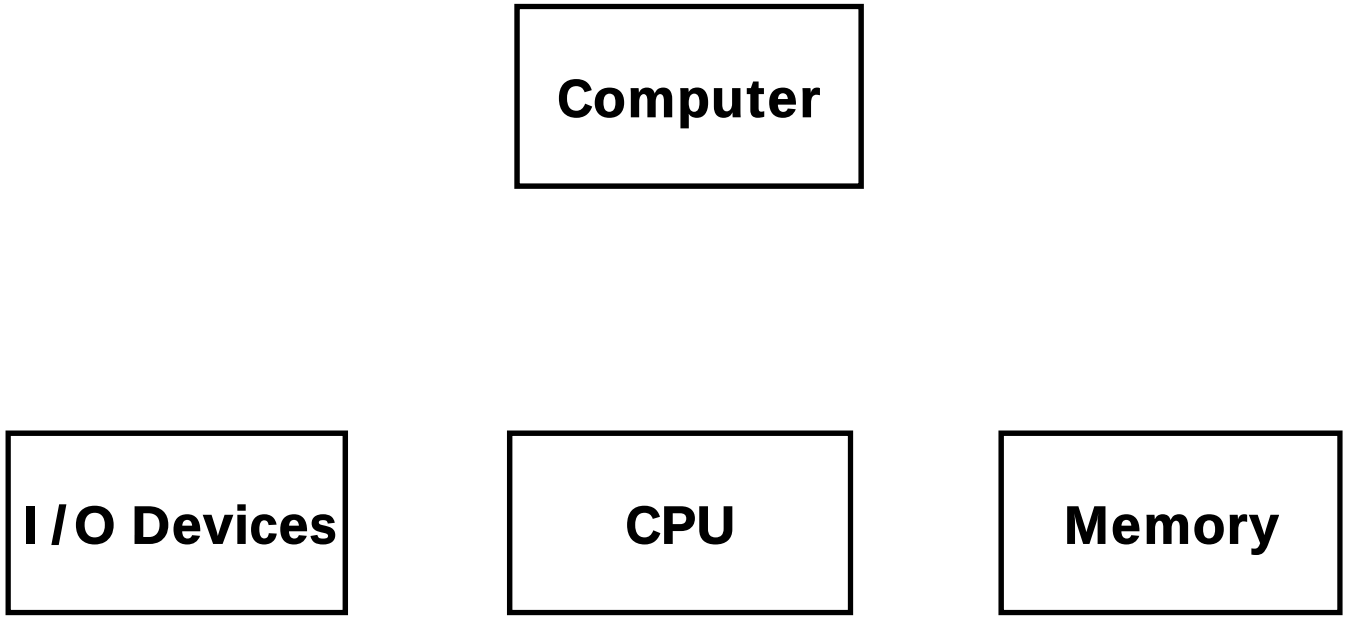
- A technique used to master complexity (“divide and conquer”)
- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major processing step (function) in the application domain
  - Modules can be decomposed into smaller modules
- Object-oriented decomposition
  - The system is decomposed into classes (“objects”)
  - Each class is a major abstraction in the application domain
  - Classes can be decomposed into smaller classes

Which decomposition is the right one?

Decomposition:
Overview

- Both views are important during software life-cycle
- Functional decomposition emphasizes the ordering of operations
  - Very useful at requirements engineering stage and high level description of the system.
  - Functions are spread over the system → Hard to maintain / change
- Object-oriented decomposition emphasizes the objects that cause the operations.
  - Very useful after initial functional description → Object Design
  - Encapsulates data and functions → helps to deal with change

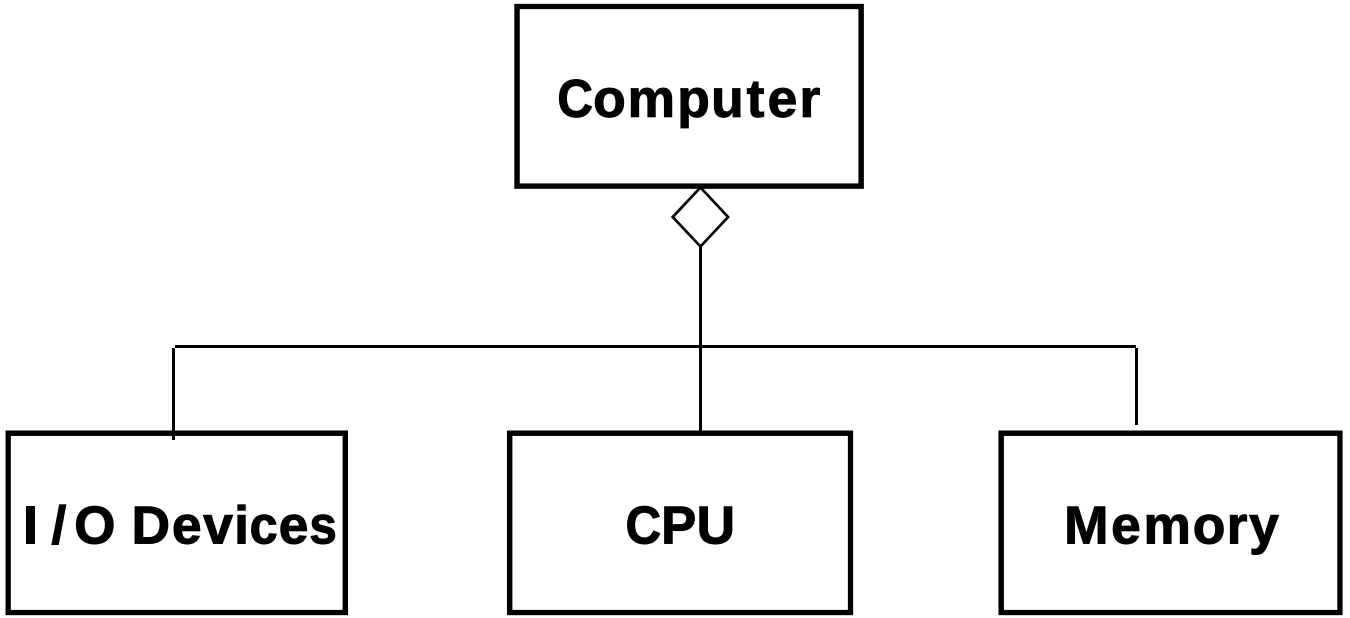
Decomposition:
Object-Oriented Decomposition



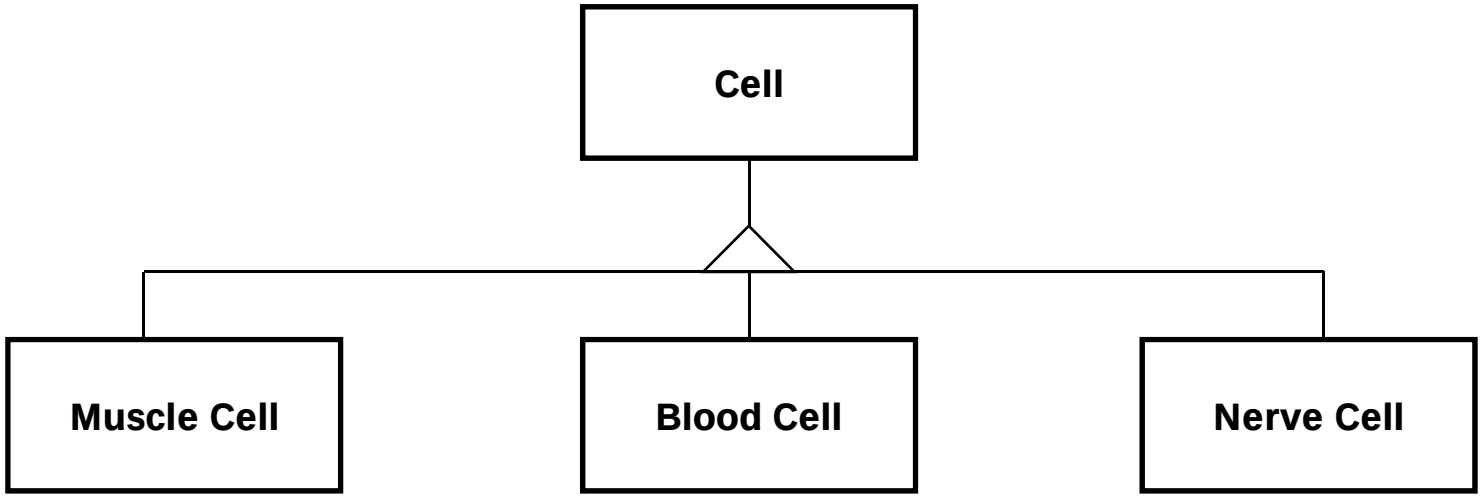
Hierarchy

- We got abstractions and decomposition
  - This leads us to chunks (classes, objects) which we view with object model
- Another way to deal with complexity is to provide simple relationships between the chunks
- One of the most important relationships is hierarchy
- 2 important hierarchies
  - "Part of" hierarchy
  - "Is-kind-of" hierarchy

Hierarchy:
Part-of-Hierarchy



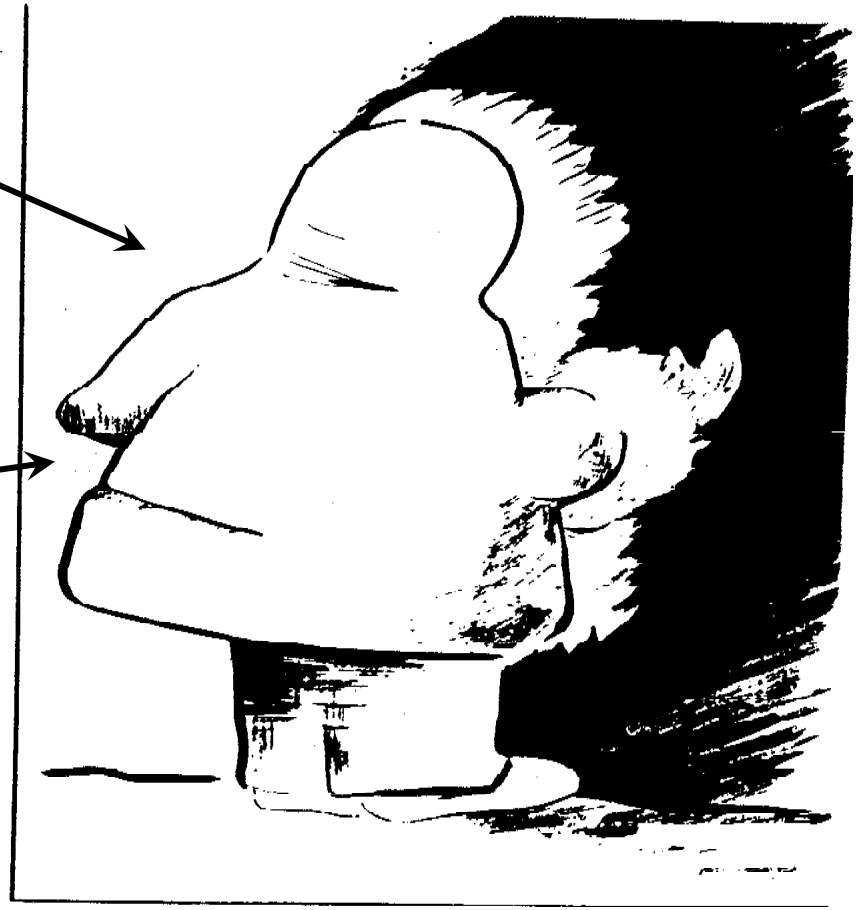
Hierarchy:
Is-Kind-of Hierarchy (Taxonomy)



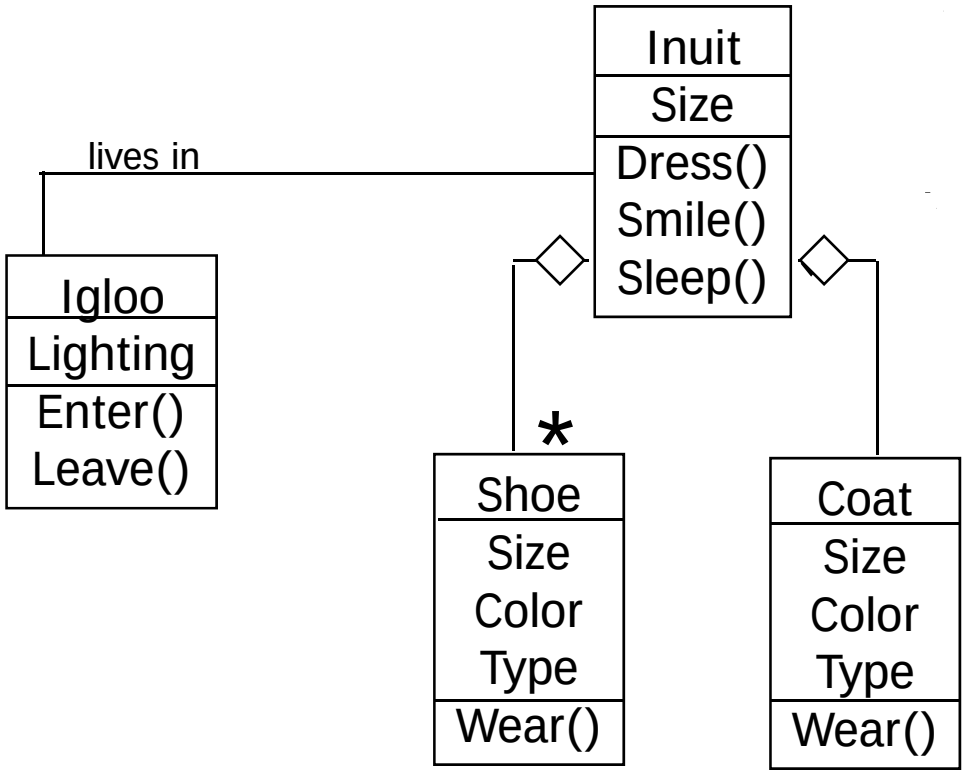
Object-oriented Modeling
Our first modeling project!

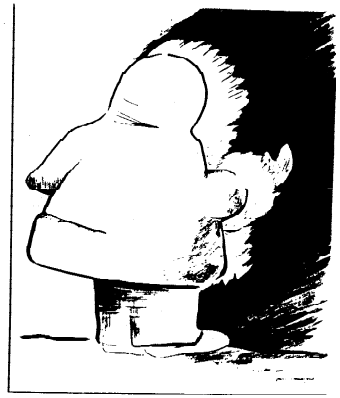
Your mission:
Please model “this”

Your (obvious) question:
What is “this”?



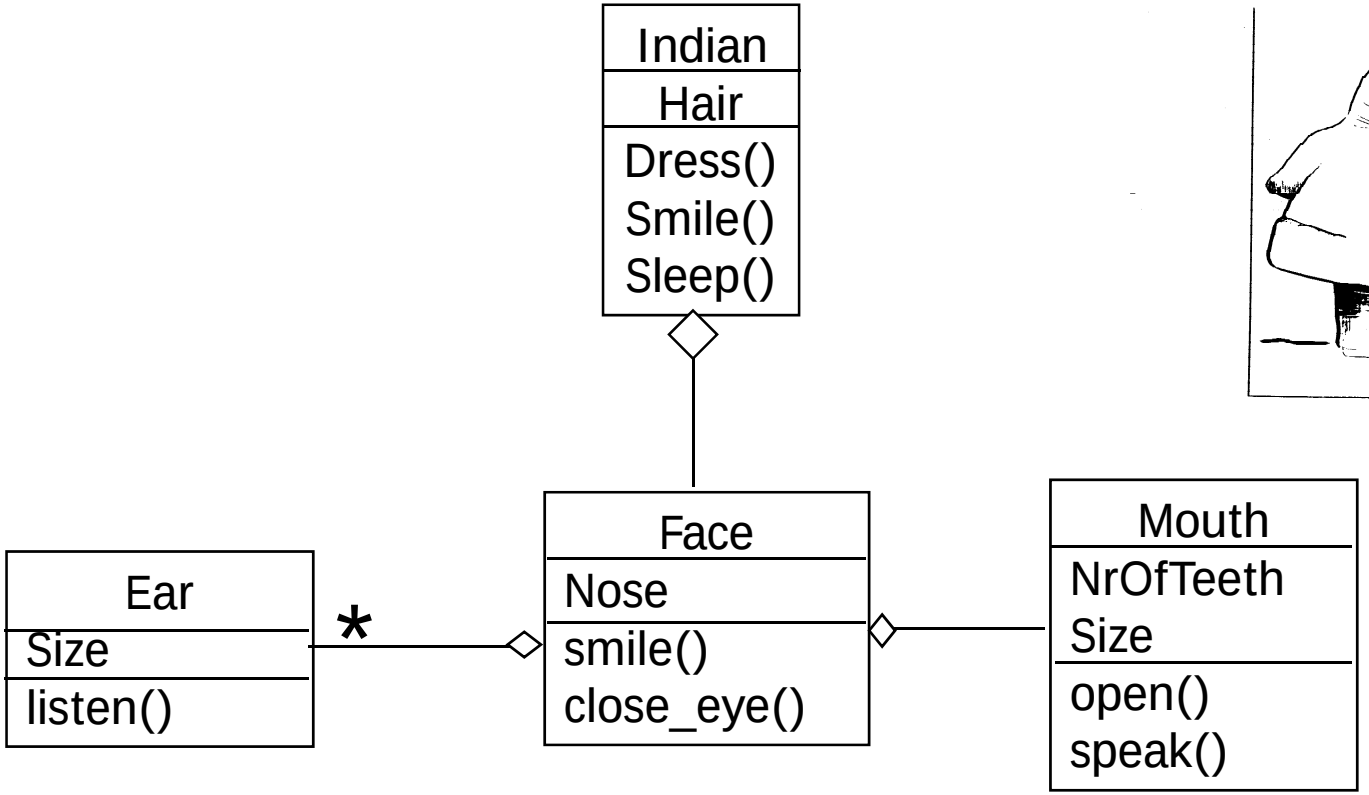
Object-oriented Modeling:
Model of an Inuit

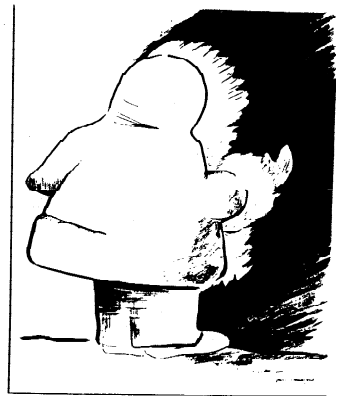




→ but is it the right model?

Object-Oriented Modeling:
Alternative Model: The Head of an Indian





Decomposition:
Class Identification

- Class identification is crucial to object-oriented modeling
- What are the limitations? Depending on the purpose of the system different objects might be found
  - How can we identify the purpose of a system?

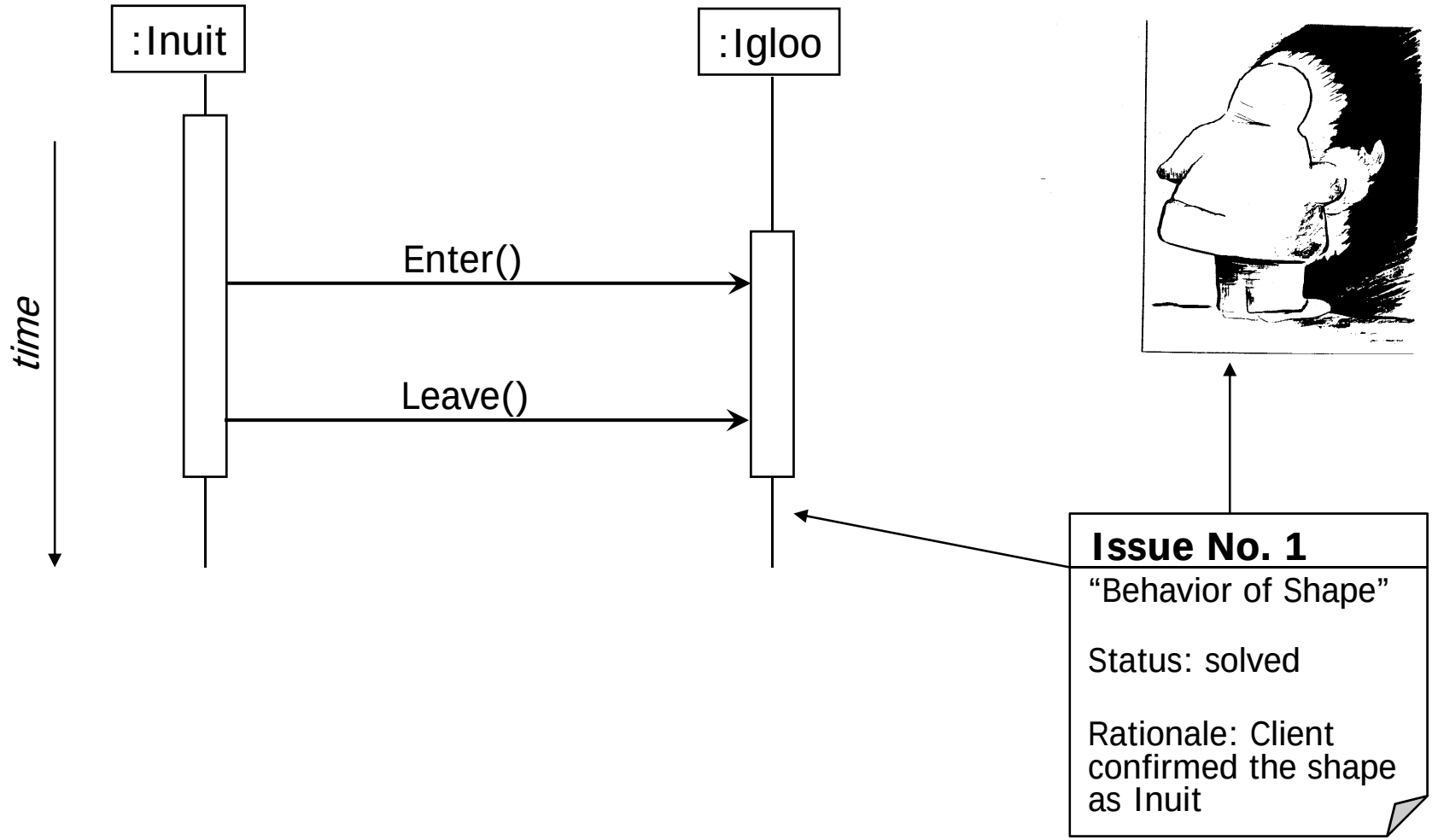
→ Principle Questions in **Requirements Engineering**

Solution:

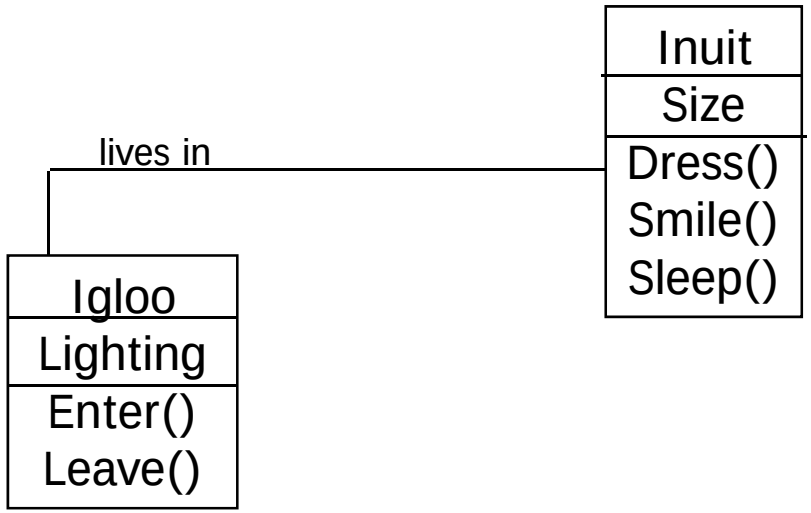
- Negotiate the purpose of a system with clients
- Present various models
- Interview, Observations ...

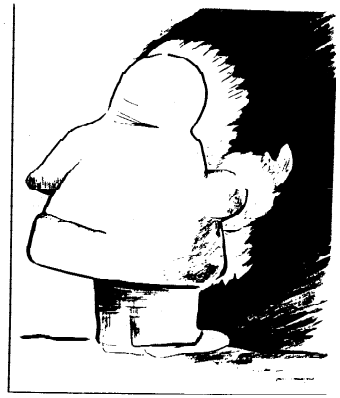
- ... ok, but what to do if the client is unknowing?

Object-oriented Modeling:
Behavior of an Inuit entering a igloo

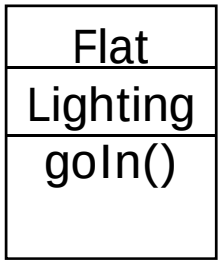


Object-oriented Modeling:
Anticipate changes





Change Request: Inuit wants to live in a flat



Goal:
Find a design in which such a change can be realized

Use already tested designs!

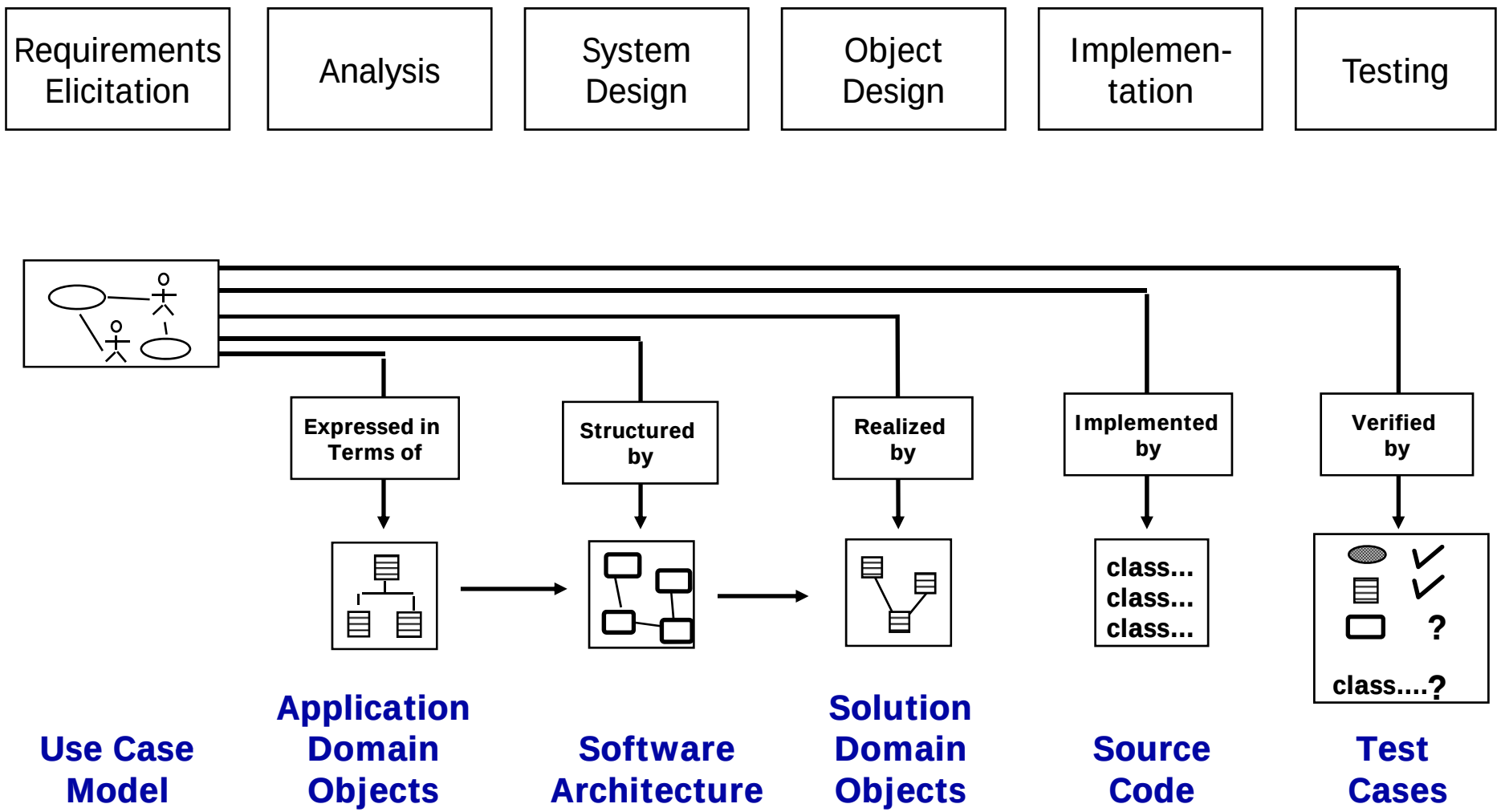
Class Identification

- Class identification is crucial to object-oriented modeling
- Basic assumption:
  1. We can find the classes for a new software system: We call this Greenfield Engineering
  2. We can identify the classes in an existing system: We call this Reengineering
  3. We can create a class-based interface to any system: We call this Interface Engineering
- Why can we do this? Philosophy, science, experimental evidence
- What are the limitations? Depending on the purpose of the system different objects might be found
  - How can we identify the purpose of a system?

So where are we right now?

- Three ways to deal with complexity and changes
  - Abstraction
  - Decomposition
  - Hierarchy
  - Anticipate changes in design
- Object-oriented modeling is a good methodology
  - Unfortunately, depending on the purpose of the system, different objects may be found
- How can we do it right?
  - Many different possibilities
  - Our current approach: Start with a description of the functionality (Use case model), then proceed to the object model
  - This leads us to the software lifecycle

Software Lifecycle Activities
... and their models



Software Lifecycle Definition

- Software lifecycle
  - Set of activities and their relationships to each other to support the development of a software system
- Typical lifecycle questions
  - Which activities should I select for the software project?
  - What are the dependencies between activities?
  - How should I schedule the activities?
- Dealing with change
  - Use a nonlinear software lifecycle to deal with changing requirements or changing technology → **Iteration**
  - Provide configuration management to deal with changing entities

Summary

- Software engineering is a problem solving activity
  - Developing quality software for a complex problem within a limited time while things are changing
  - Object-Oriented Software Construction is the standard method
- There are many ways to deal with complexity
  - Modeling, decomposition, abstraction, hierarchy
- Many ways to do deal with change
  - **Management View:** Use of an iterative software lifecycle
  - **Technical View:** Anticipate changes in your design (→ use design patterns)

Overview on Lectures

- Introduction
- Warm up tutorial: OO, Java, Eclipse, SVN
- UML
- Requirements Elicitation
- Requirements Analysis
- **System Design**
- **Object Design**
  - Design Patterns
  - Refactoring
- Testing
- Configurations Management
- Rationale Management
- Special lectures on selected topics (t.b.a.)
- Practice Talks (t.b.a)

Lecture Organization

Class Organization
Questionnaire + Grouping

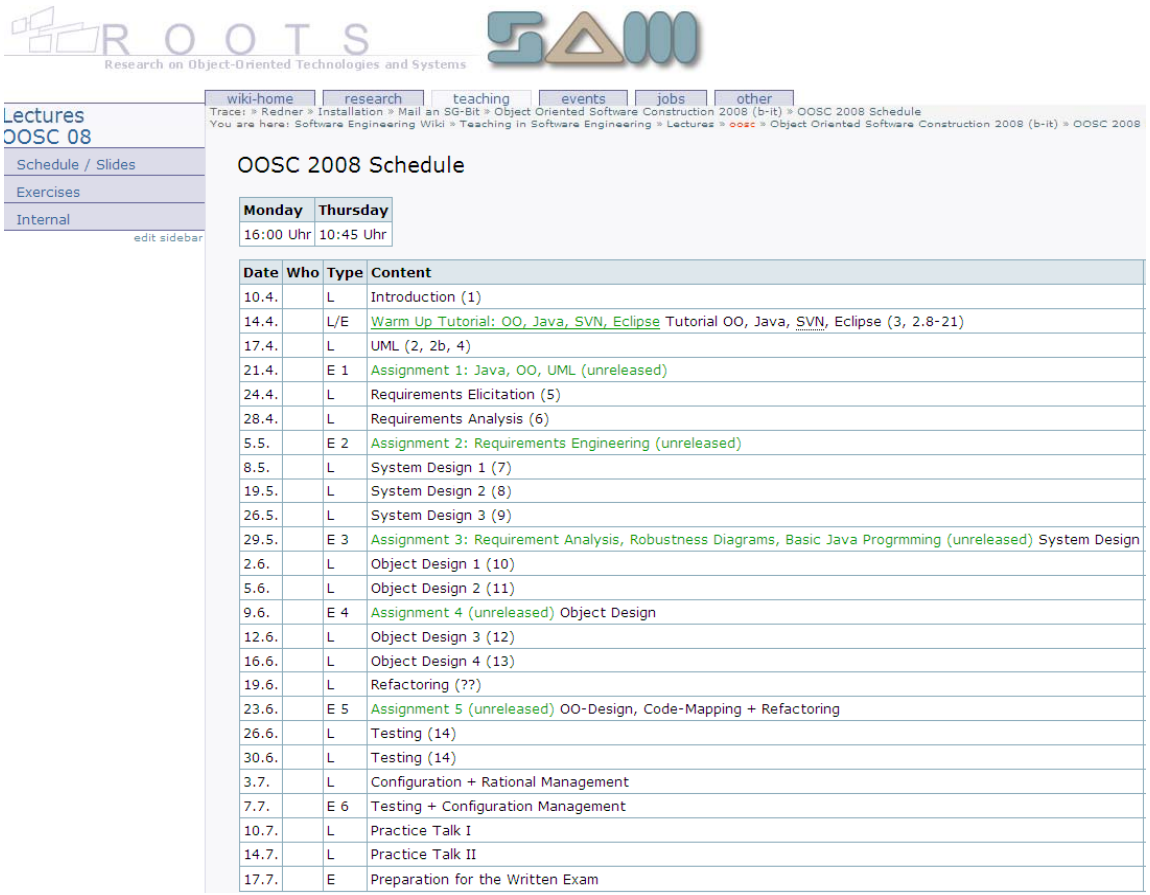
- Please fill in the questionnaire and join a group
  - You will be added to the OOSC mailing list and will
  - get a SVN account
- Questionnaire
  - Name
  - Email address
  - Bachelor's degree
  - Knowledge in Eclipse
  - Knowledge in Java
  - Programming Languages
  - Knowledge in UML
  - Knowledge in SVN (Version Control System)
  - **Exercise Grouping Number**

Class Organization
Dates + Mode

- Dates:
  - Monday, 17:00 – 18:30 Great Lecture Hall, B-IT
  - Thursday, 10:35 – 12:05 Great Lecture Hall, B-IT
- Lecture Mode:
  - every first week: two lectures
  - every second week: one exercise, one lecture

Lecture Organization

- Webpage
  https://sewiki.iai.uni-bonn.de/teaching/lectures/oosc/2008/start
- Mailing-List
  oosc-lecture@iai.uni-bonn.de
- Slides for lecture are published on webpage approx. 1-2 hours before the lecture takes place



ROOTS
Research on Object-Oriented Technologies and Systems

wiki-home | research | teaching | events | jobs | other

Trace: » Redner » Installation » Mail an SG-Bit » Object Oriented Software Construction 2008 (b-it) » OOSC 2008 Schedule
You are here: Software Engineering Wiki » Teaching in Software Engineering » Lectures » oosc » Object Oriented Software Construction 2008 (b-it) » OOSC 2008

Lectures
OOSC 08
- Schedule / Slides
- Exercises
- Internal

edit sidebar

OOSC 2008 Schedule

Monday	Thursday
16:00 Uhr	10:45 Uhr

Date	Who	Type	Content
10.4.		L	Introduction (1)
14.4.		L/E	Warm Up Tutorial: OO, Java, SVN, Eclipse Tutorial OO, Java, SVN, Eclipse (3, 2.8-21)
17.4.		L	UML (2, 2b, 4)
21.4.		E 1	Assignment 1: Java, OO, UML (unreleased)
24.4.		L	Requirements Elicitation (5)
28.4.		L	Requirements Analysis (6)
5.5.		E 2	Assignment 2: Requirements Engineering (unreleased)
8.5.		L	System Design 1 (7)
19.5.		L	System Design 2 (8)
26.5.		L	System Design 3 (9)
29.5.		E 3	Assignment 3: Requirement Analysis, Robustness Diagrams, Basic Java Progrmming (unreleased) System Design
2.6.		L	Object Design 1 (10)
5.6.		L	Object Design 2 (11)
9.6.		E 4	Assignment 4 (unreleased) Object Design
12.6.		L	Object Design 3 (12)
16.6.		L	Object Design 4 (13)
19.6.		L	Refactoring (??)
23.6.		E 5	Assignment 5 (unreleased) OO-Design, Code-Mapping + Refactoring
26.6.		L	Testing (14)
30.6.		L	Testing (14)
3.7.		L	Configuration + Rational Management
7.7.		E 6	Testing + Configuration Management
10.7.		L	Practice Talk I
14.7.		L	Practice Talk II
17.7.		E	Preparation for the Written Exam

Class Organization
Requirements to be credited

- SWS: V3/Ü1, number of ECTS Credits: 6
- Requirements to get admittance to the written exam
  - Join an exercise group of up to **four students**
  - **50 %** of all assignments must be completed by the group
  - Assignments are reviewed and graded
  - Regular attendance to the exercises by at least one group member
  - Present at least one solution in the exercises
- Bonus points for the exam
  - The exam grade is **reduced by 0,3** points if your group solves >= **70 %**
- Written exam
  - Date to be announced soon: presumably in the week 21.-25.07.08
  - Second date (oral or written) will be announced shortly after the first exam
  - Covered: Everything that
    - ➔ is discussed in the lectures
    - ➔ is elaborated in the exercises

Class Organization
Exercise Announcement and Submission

- Assignments will be announced on the website one week before the next exercise
  - Also published in the mailing list
- Solutions have to be prepared for the exercise *in groups (max. 4 students)*
  - Preparation of the assignments with tools (Eclipse/SVN, JUDE/UML) or text editors
  - Due: typically two days before the exercise session starts, **10:45 am** to Mahmoud El-Gayar (**elgayyar@iai.uni-bonn.de**), or the version control system (SVN).
    Actual dates will be announced with every assignment
  - Fill in the group information form or send initial group information (title, names, email addresses, and matrikel number of students) to Mahmoud El-Gayar until April, 14th 2007, 12:00

Lecture Organization
Exercise Dates

- 17 lecture sessions (April 10rd until July 14th)
  - 1 – 2 practice talks in July
- 7 exercise sessions
- Exercise dates:
  - ➔ Monday, April, 14th 2008 (OO/Java/Eclipse Tutorial)
  - ➔ Monday, May, 5th 2008
  - ➔ Thursday, May, 29th 2008
  - ➔ Monday, June, 9th 2008
  - ➔ Monday, June, 23th 2008
  - ➔ Monday, July, 7th 2008
  - ➔ Thursday, July, 17th 2008
- Optional Java Tutorials
  - ➔ Wednedays, upon consultation

Warm Up Tutorial
OO, Java, SVN, Eclipse

- Monday, April 14th
- Short Introductions
  - Object-Orientated Modeling
  - Java
  - Eclipse
  - SVN
    - ➔ The version control system we use to organize the assignments
- Bring your Laptops and set up your environment for the assignments beforehand
  - At least one laptop per group
    - ➔ If this is not possible please contact us directly or via the mailing list
  - https://sewiki.iai.uni-bonn.de/teaching/lectures/oosc/2008/exercises/warmup/start
  - Make sure to set up your environment before the tutorial starts