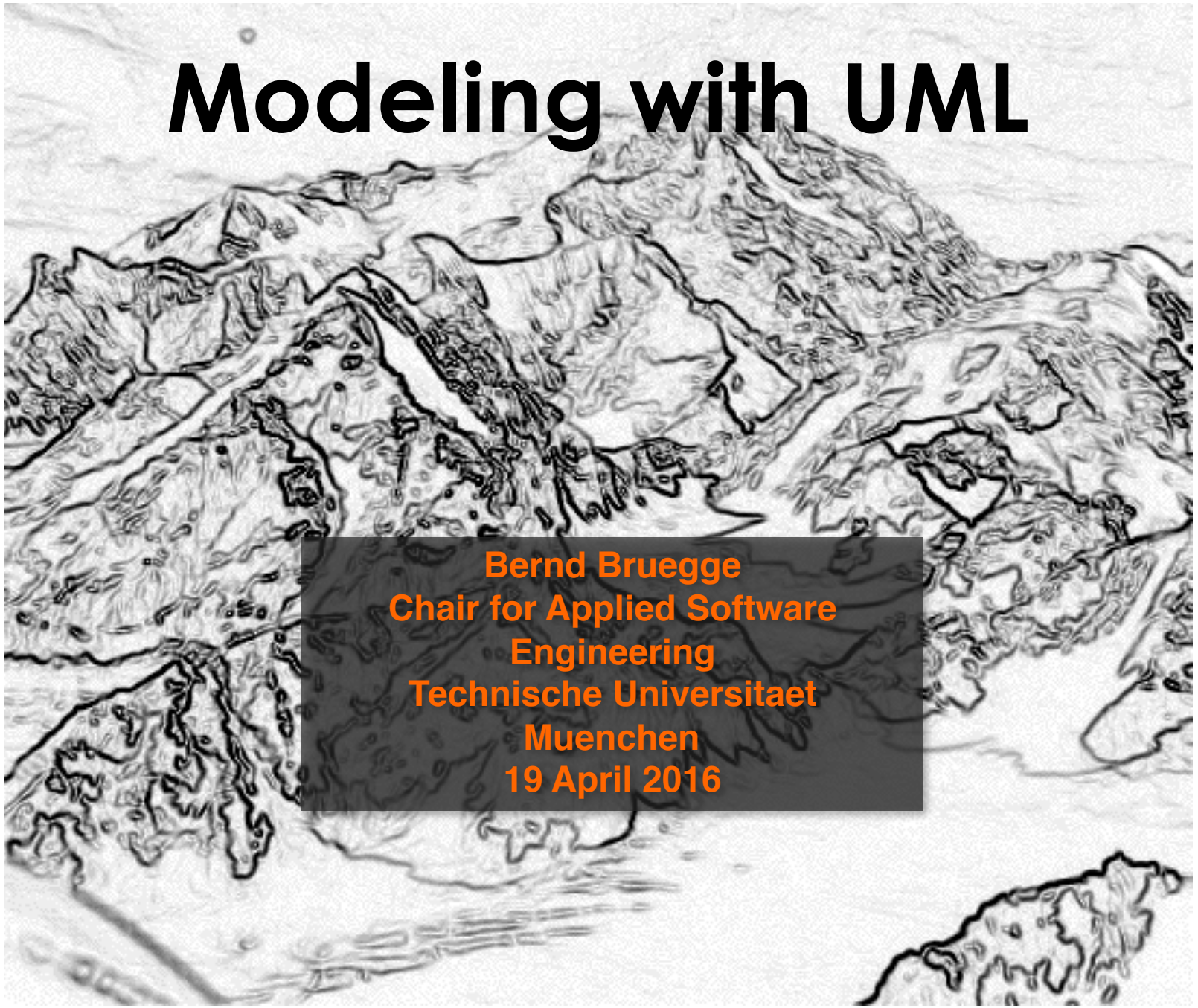
Introduction into Software Engineering

Modeling with UML



Bernd Bruegge
Chair for Applied Software Engineering
Technische Universitaet Muenchen
19 April 2016

Overview for Today's Lecture

- Odds and Ends (Changes)
  - Moodle Course Availability and Live streaming
  - Changes to the EIST Schedule
- Morning Quiz
- Three ways to deal with complexity
  - ✓ Abstraction and Modeling
  - Decomposition
  - Hierarchy
- Introduction into the UML notation. First Pass on:
  - Use case diagrams
  - Class diagrams
  - Statechart diagrams
  - Communication diagrams
  - Sequence diagrams

Moodle Course Availability

- A Moodle course has been created for EIST:
- https://www.moodle.tum.de/course/view.php?id=28495

- Moodle contains all the information you need in one location:
  - Announcements
  - Lecture schedule
  - Lecture slides
  - Morning Quizzes
  - Exercise hand-in uploads
  - Lecture recordings (will be available asap)

Livestream.com – Live streaming

- This course will be live streamed to:
- Livestream.com
- URL: http://livestream.com/accounts/1073403/eist2016/

- Or just search for "EIST" @ livestream.com

Changes to the EIST Schedule

Lecture	Day	Date	Subject	Time
1	Thu	14.04.2016	Introduction	08:00-10:15
2	Tue	19.04.2016	Modeling with UML #1	12:15-14:00
3	Thu	21.04.2016	Modeling with UML #2	08:00-10:15
4	Tue	26.04.2016	Requirements Elicitation	12:15-14:00
~~5~~	~~Thu~~	28.04.2016	System Modeling	08:00-10:15
6	Tue	03.05.2016	Design Patterns I and II	12:15-14:00
7	Thu	05.05.2016	Reverse Engineering	08:00-10:15
8	Tue	10.05.2016	Detailed Design	12:15-14:00
9	Thu	12.05.2016	System Design I	08:00-10:15
~~10~~	~~Tue~~	17.05.2016	System Design II	12:15-14:00
11	Thu	19.05.2016	Quality Management I	08:00-10:15
12	Tue	24.05.2016	Quality Management II	12:15-14:00
~~13~~	~~Thu~~	26.05.2016	Interface Specification	08:00-10:15
14	Tue	31.05.2016	Model Transformations I	12:15-14:00
15	Thu	02.06.2016	Model Transformations II	08:00-10:15
16	Tue	07.06.2016	Lifecycle Modeling I	12:15-14:00

Changes to the EIST Schedule

Lecture	Day	Date	Subject	Time
1	Thu	14.04.2016	Introduction	08:00-10:15
2	Tue	19.04.2016	Modeling with UML #1	12:15-14:00
3	Thu	21.04.2016	Modeling with UML #2	08:00-10:15
4	Tue	26.04.2016	Requirements Elicitation	12:15-14:00
~~5~~	~~Thu~~	28.04.2016	No Lecture	08:00-10:15
6	Tue	03.05.2016	Design Patterns I and II	12:15-14:00
7	Thu	05.05.2016	Reverse Engineering	08:00-10:15
8	Tue	10.05.2016	Detailed Design	12:15-14:00
9	Thu	12.05.2016	System Design I	08:00-10:15
~~10~~	~~Tue~~	17.05.2016	No Lecture	12:15-14:00
11	Thu	19.05.2016	Quality Management I	08:00-10:15
12	Tue	24.05.2016	Quality Management II	12:15-14:00
~~13~~	~~Thu~~	26.05.2016	No Lecture	08:00-10:15
14	Tue	31.05.2016	Model Transformations I	12:15-14:00
15	Thu	02.06.2016	Model Transformations II	08:00-10:15
16	Tue	07.06.2016	Lifecycle Modeling I	12:15-14:00

Updated Course Schedule (1)

Lecture	Day	Date	Subject	Time
1	Thu	14.04.2016	Introduction	08:00-10:15
2	Tue	19.04.2016	Modeling with UML #1	12:15-14:00
3	Thu	21.04.2016	Modeling with UML #2	08:00-10:15
4	Tue	26.04.2016	Requirements Ecliciation	12:15-14:00
5	Thu	28.04.2016	System Modeling	08:00-10:15
6	Tue	03.05.2016	Design Patterns I and II	12:15-14:00
7	Thu	05.05.2016	No Lecture	08:00-10:15
8	Tue	10.05.2016	Object Design	12:15-14:00
9	Thu	12.05.2016	System Design I	08:00-10:15
10	Tue	17.05.2016	No Lecture	12:15-14:00
11	Thu	19.05.2016	System Design II	08:00-10:15
12	Tue	24.05.2016	Testing I	12:15-14:00
13	Thu	26.05.2016	No Lecture	08:00-10:15
14	Tue	31.05.2016	Testing II	12:15-14:00
15	Thu	02.06.2016	Interface Specification	08:00-10:15
16	Tue	07.06.2016	Model Transformations I	12:15-14:00

Updated Course Schedule (2)

Lecture	Day	Date	Subject	Time
17	Thu	09.06.2016	No Lecture	08:00-10:15
18	Tue	14.06.2016	No Lecture	12:15-14:00
19	Thu	16.06.2016	Model Transformations II	08:00-10:15
29	Tue	21.06.2016	Lifecycle Modeling I	12:15-14:00
21	Thu	23.06.2016	Lifecycle Modeling II	08:00-10:15
22	Tue	28.06.2016	Mapping UML	12:15-14:00
23	Thu	30.06.2016	Presentation of Large complex system	08:00-10:15
24	Tue	05.07.2016	Project Management	12:15-14:00
25	Thu	07.07.2016	Release Management	08:00-10:15
26	Tue	12.07.2016	System Maintenance & Evolution	12:15-14:00
27	Thu	14.07.2016	Wrapup and Exam Preparation	08:00-10:15

- Final exam: 28 July 2016
- Repeat exam 10 October 2016
- Note: There will be no midterm.

Overview for Today's Lecture

- Odds and Ends (Changes)
  - ✓ Moodle Course Availability
  - ✓ Changes to the EIST Schedule
- Morning Quiz
- Three ways to deal with complexity
  - Abstraction and Modeling
  - Decomposition
  - Hierarchy
- Introduction into the UML notation. First Pass on:
  - Use case diagrams
  - Class diagrams
  - Statechart diagrams
  - Communication diagrams
  - Sequence diagrams

Changes to the Grading Criteria

- To pass this course, your exam grade must be 4.0 or better
- There will be no bonus
- For each exercise:
  - we determine the 3 fastest and correct submissions
  - we do a random drawing of all submitted exercises.

- Details on the in-class exercises
  - There will be up to 3 exercises per class
  - Each exercise is time-bound

Morning Quiz

- At the beginning of each lecture you can participate in the morning quiz
- Today's quiz question is available on Moodle
  - https://www.moodle.tum.de/course/view.php?id=28495

- We will offer a morning quiz at the beginning of each lecture for the rest of the semester

Overview for Today's Lecture

- Odds and Ends (Changes)
  - ✓Moodle Course Availability
  - ✓Changes to the EIST Schedule
- Morning Quiz
- Three ways to deal with complexity
  - ✓Abstraction and Modeling (last lecture)
  - Decomposition
  - Hierarchy
- Introduction into the UML notation. First Pass on:
  - Use case diagrams
  - Class diagrams
  - Statechart diagrams
  - Communication diagrams
  - Sequence diagrams

Decomposition

- A technique used to master complexity ("divide and conquer")

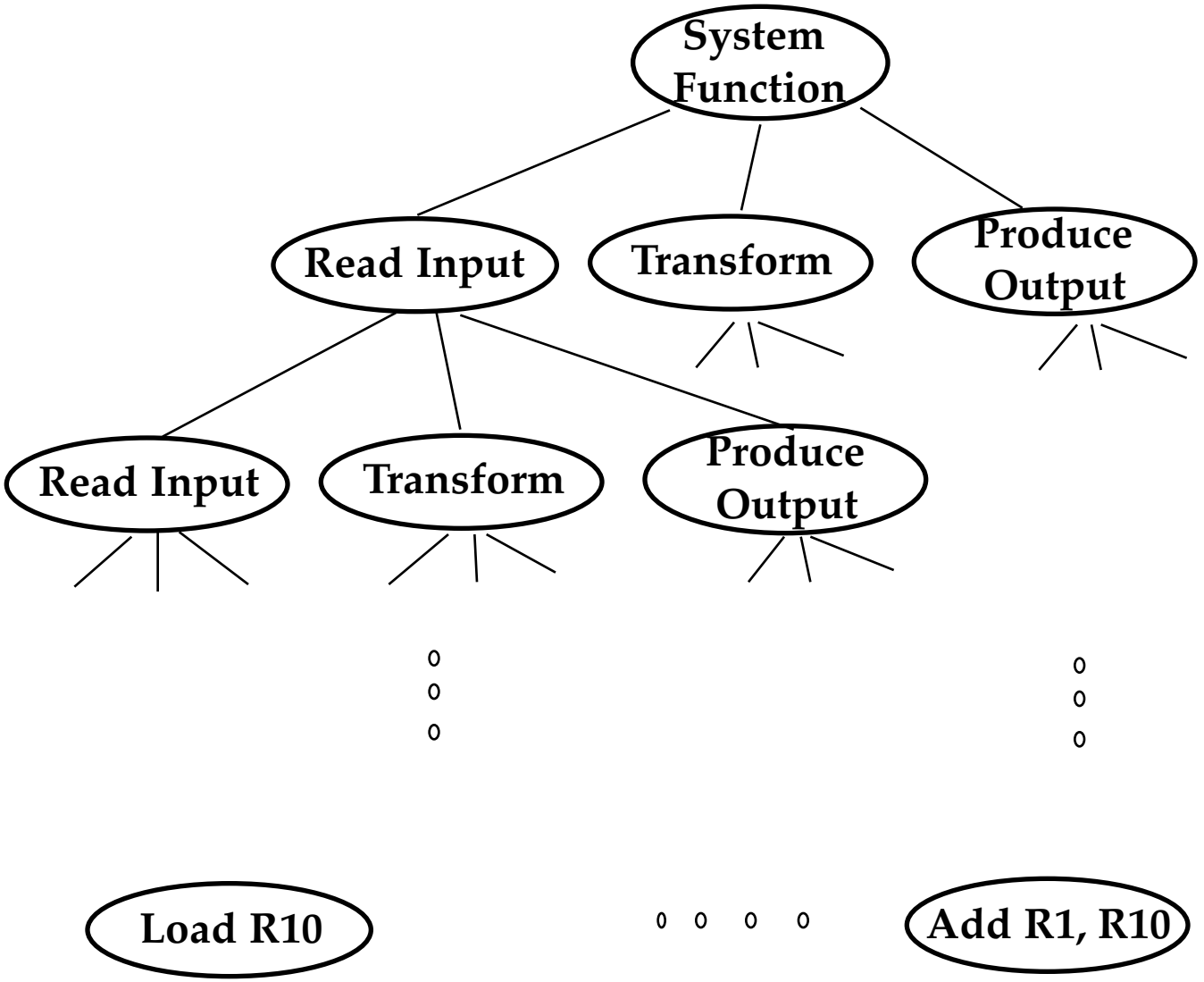
- Two major types of decomposition
  - Functional decomposition
  - Object-oriented decomposition.

Functional vs Object-oriented decomposition

- Functional decomposition
  - The system is decomposed into functions
  - Functions can be decomposed into smaller smaller functions
- Object-oriented decomposition
  - The system is decomposed into classes ("objects")
  - Classes can be decomposed into smaller classes.

Which decomposition is the right one?

Functional Decomposition



Top Level functions

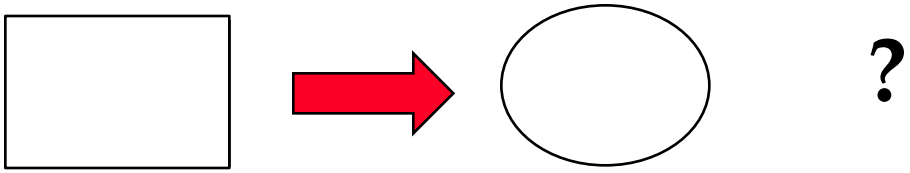
Level 1 functions

Level 2 functions

Machine instructions

Functional Decomposition

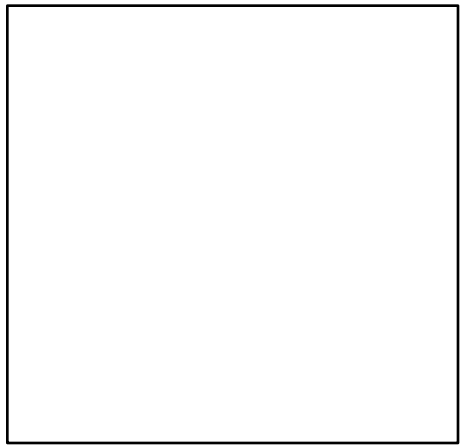
- Example: Graphics Program
  - How do I change a square into a circle?



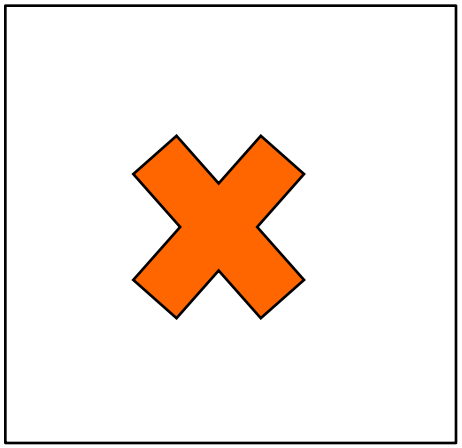
?

- Let's try this with Microsoft Powerpoint
- Let's go into Edit Mode

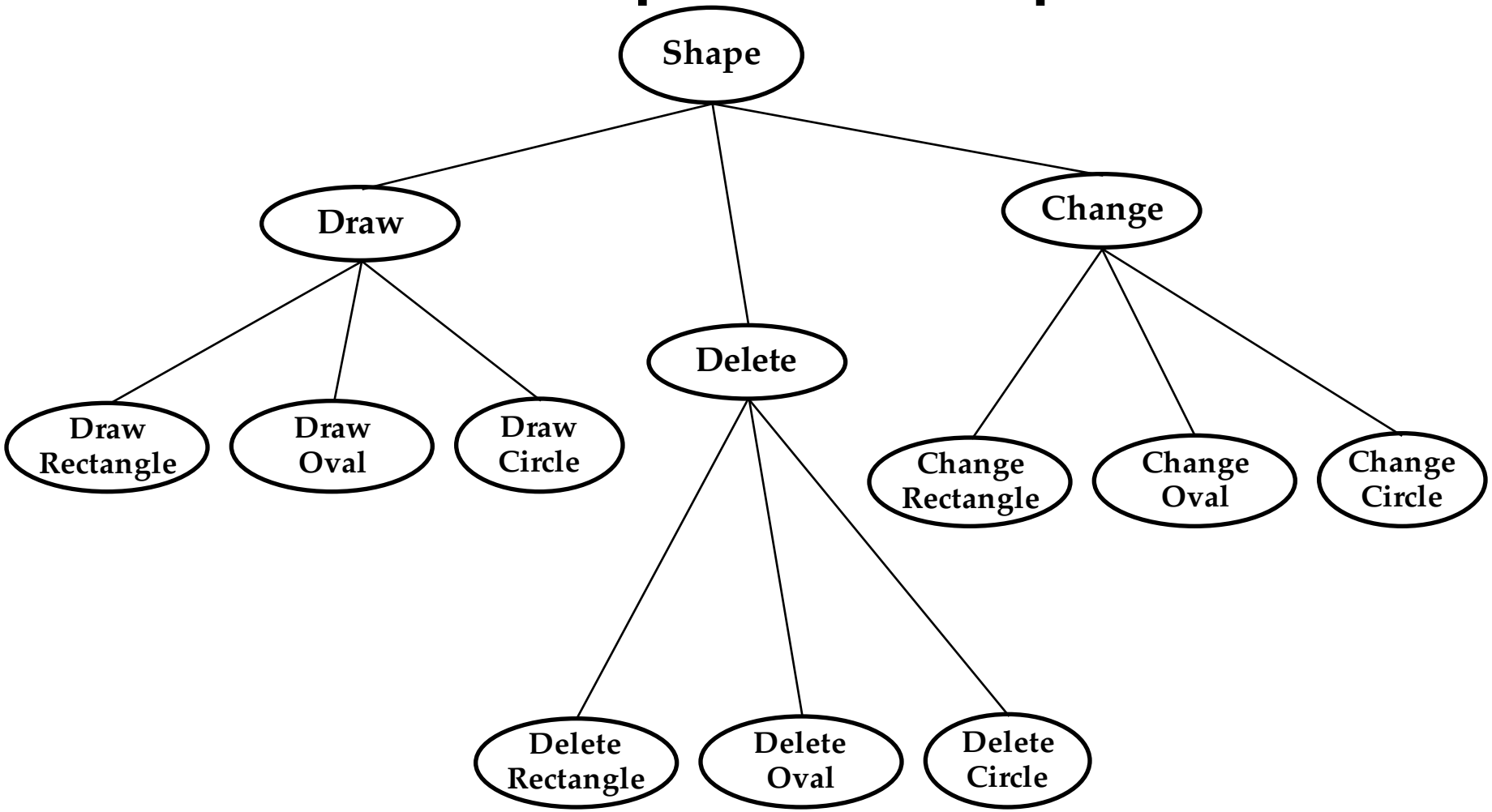
First Attempt: Right Click on Rectangle



Second Attempt☺



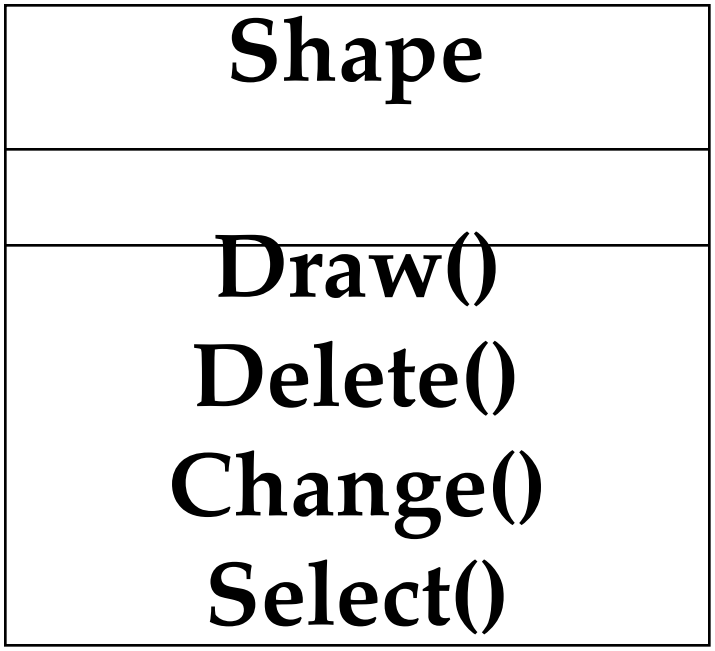
Functional Decomposition: Shape



Functional Decomposition

- The functionality is spread all over the system
  - Source code is hard to understand
  - Source code is complex and impossible to maintain
  - User interface is often awkward and non-intuitive.
- Consequence:
  - A maintainer must often understand the whole system before making a single change to the system

Object-Oriented View

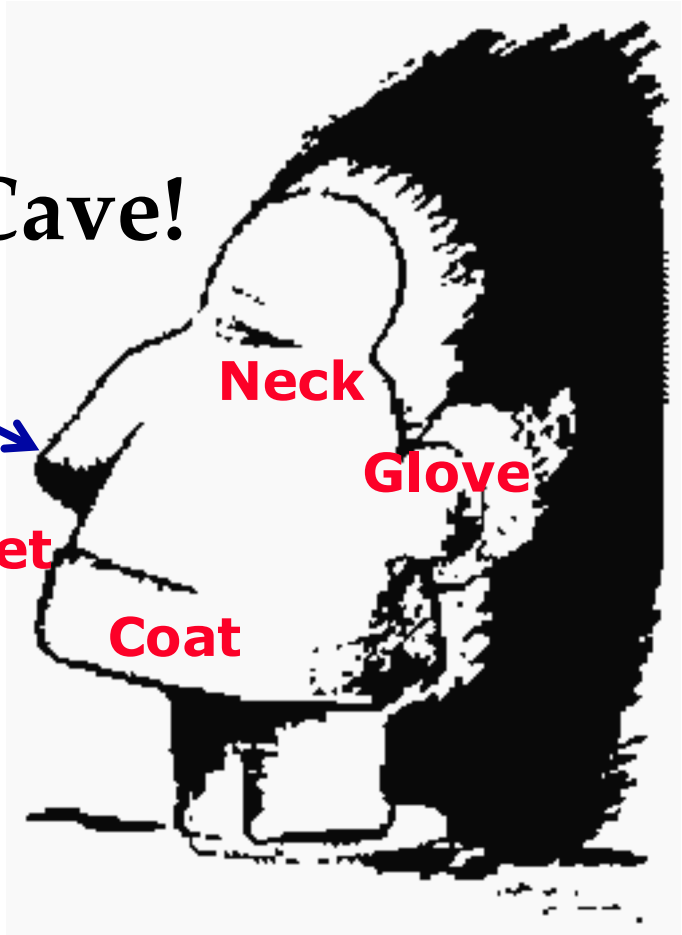


Is this really better?

What is This?

Let's try object-oriented decomposition

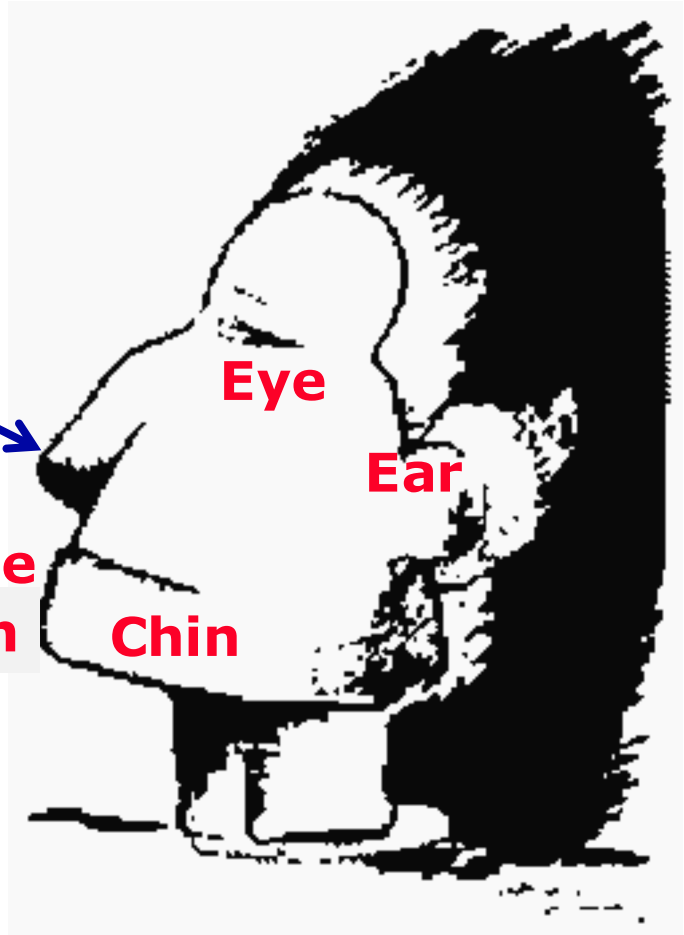
An Eskimo Entering a Cave!



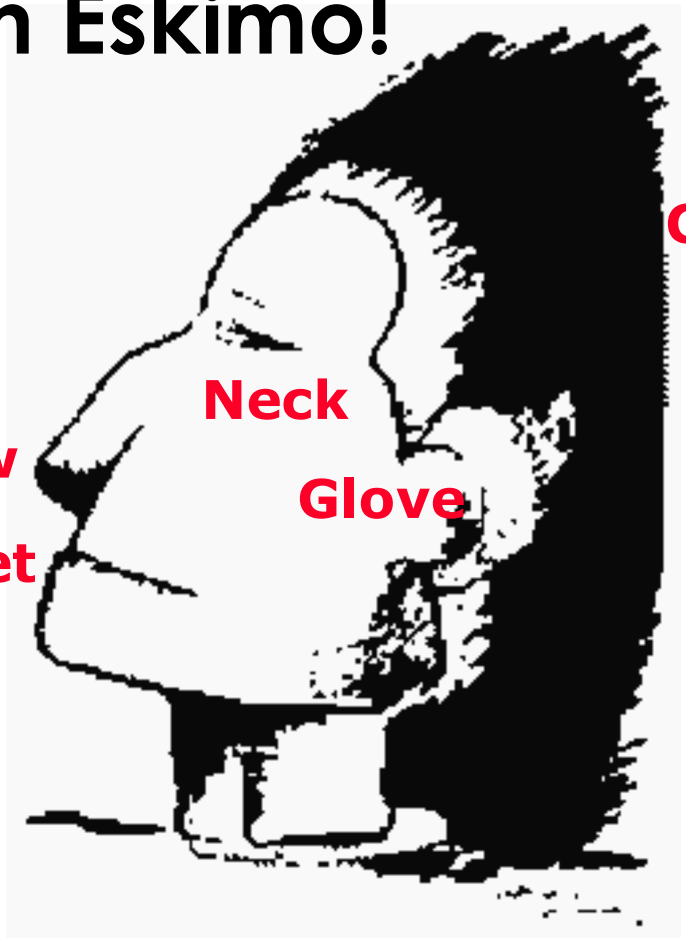
What is This?

Let's try again

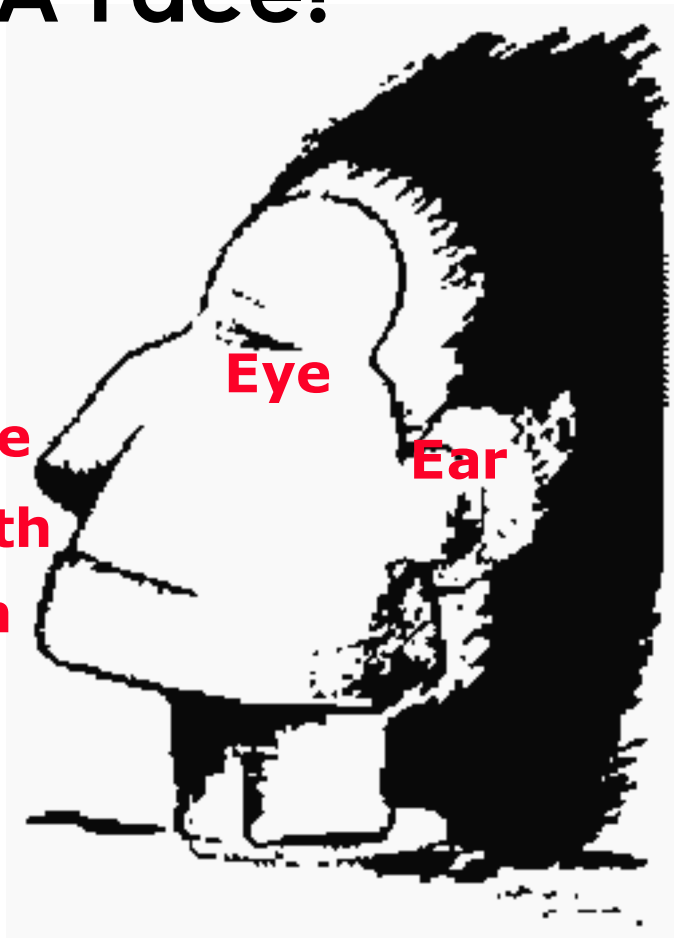
A Face!



An Eskimo!



A Face!



Important in Modeling: Find the right classes!

Class Identification

- **We can find classes:**
  - Philosophy, science, experimental evidence
- **There are limitations**:
  - Depending on the purpose of the system, different objects might be found
- **Crucial:**

  Identify the purpose of a system.

Class Identification and Project Types

- **Basic assumptions:**
  - We can find the *classes for a future system:* Greenfield Engineering Project
  - We can identify the *classes in an existing system (legacy system)*: Reengineering Project
  - We can create a *class-based interface to an existing system:* Interface Engineering Project

Modeling a system that no longer exists (Out of EIST scope).

Hierarchy

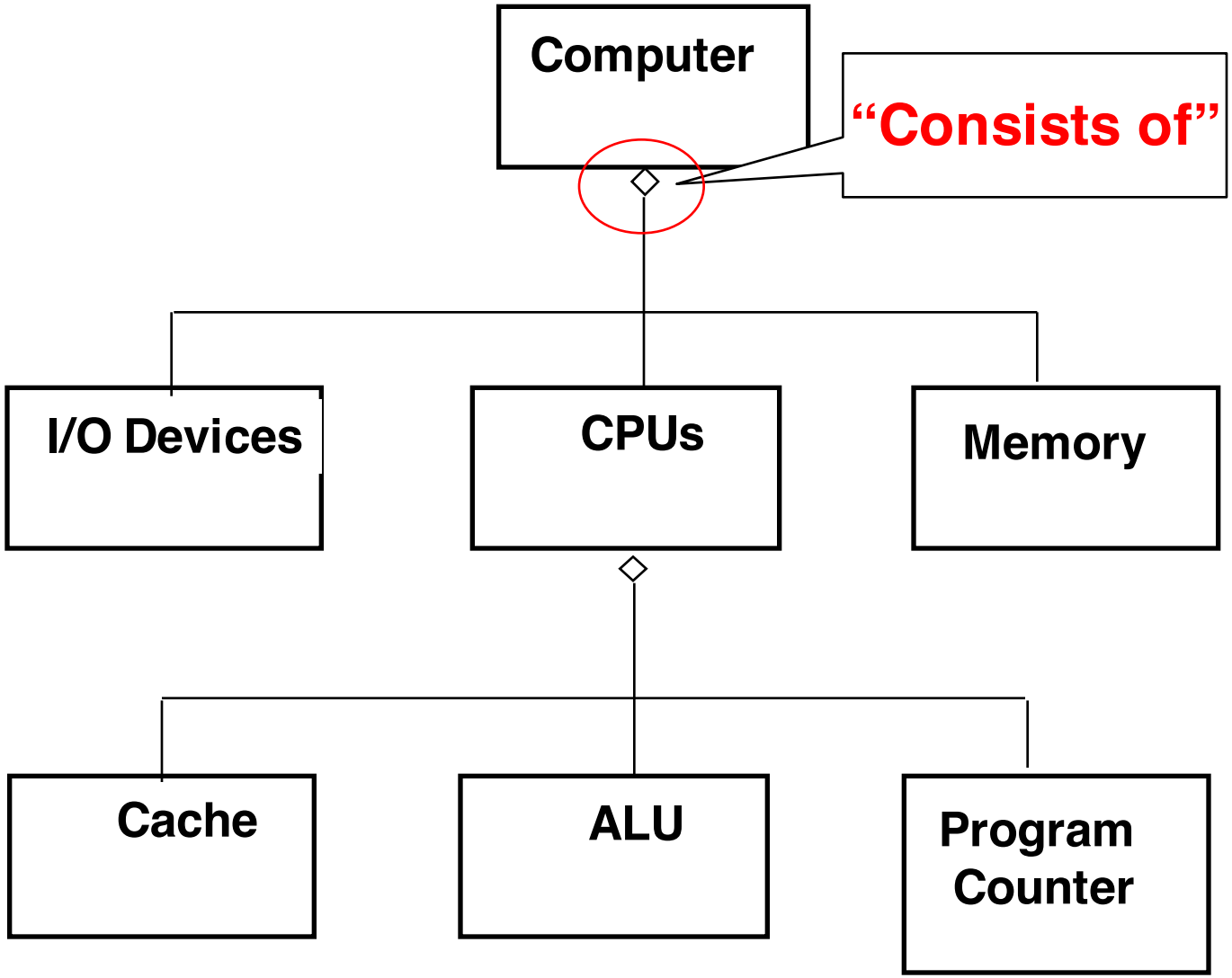
- So far we got abstractions and decomposition
  - Abstraction: Identification of classes and objects
  - Decomposition: Modules, Subsystems, Packages

- Another way to deal with complexity is to provide relationships between these chunks
- One of the most important relationships is hierarchy
- 2 special hierarchies
  - "Part-of" hierarchy
  - "Is-kind-of" hierarchy.

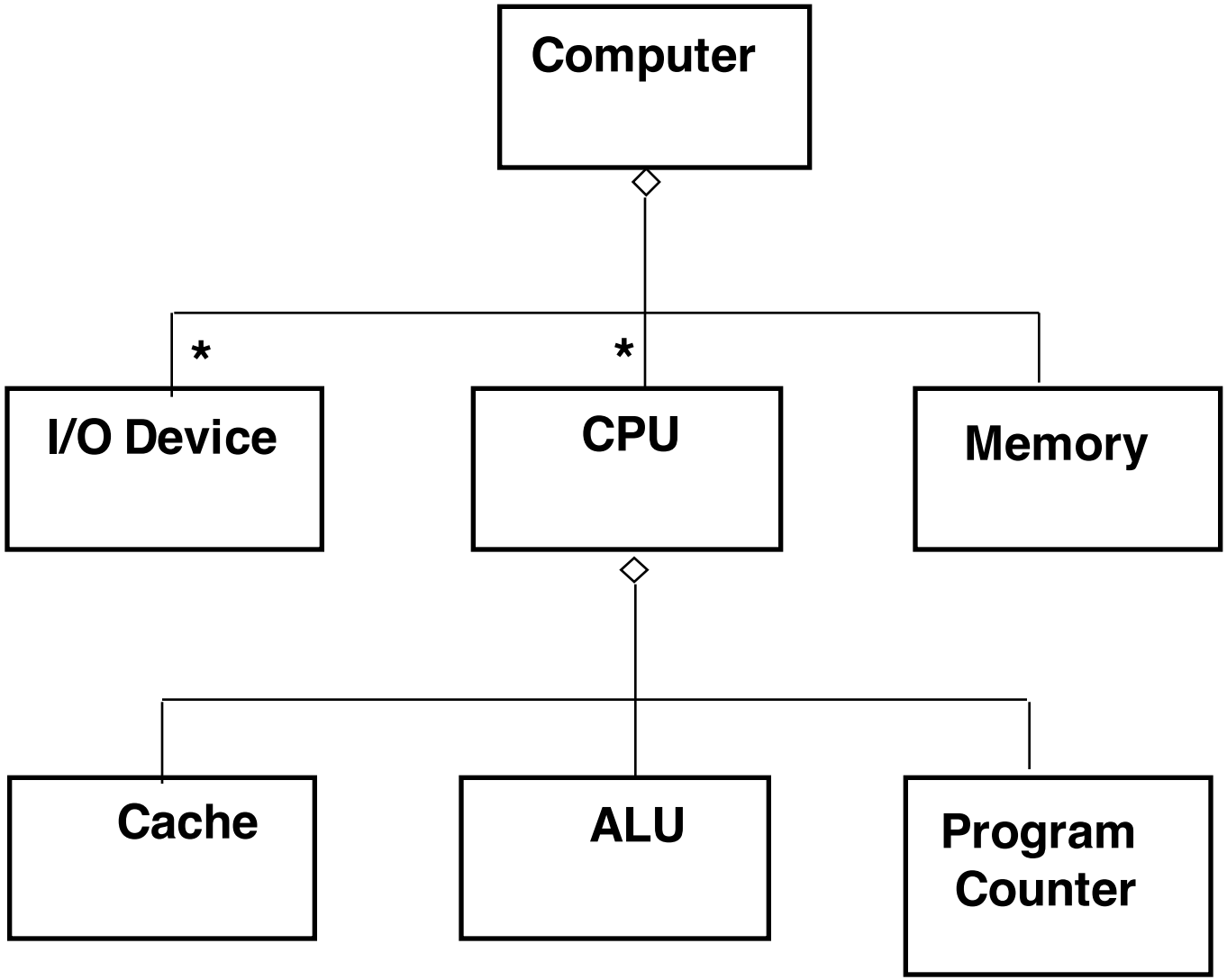
Overview for Today's Lecture

- Odds and Ends (Changes)
  - ✓ Moodle Course Availability
  - ✓ Changes to the EIST Schedule
- Morning Quiz
- Three ways to deal with complexity
  - ✓ Abstraction and Modeling (last lecture)
  - ✓ Decomposition
  - → Hierarchy
- Introduction into the UML notation. First Pass on:
  - Use case diagrams
  - Class diagrams
  - Statechart diagrams
  - Communication diagrams
  - Sequence diagrams

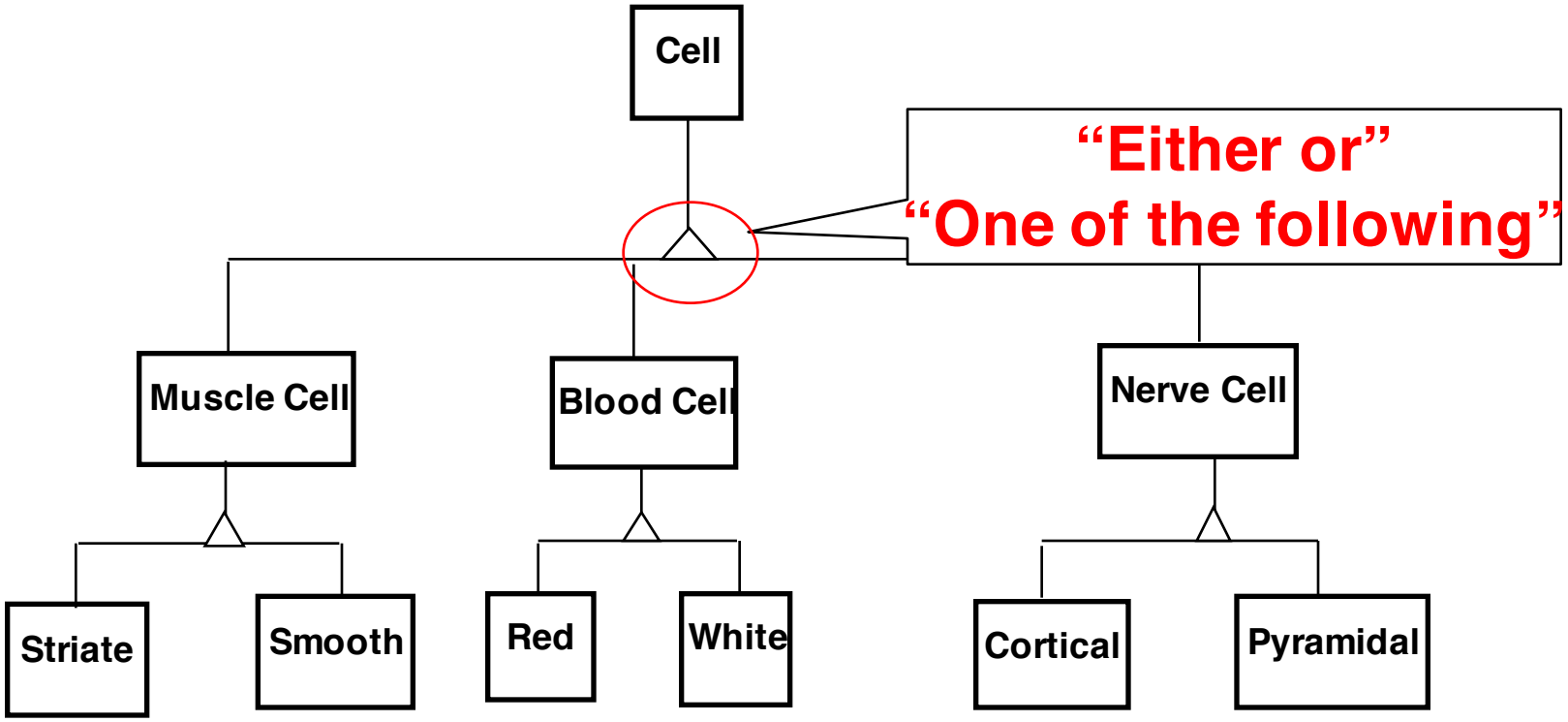
Part-of Hierarchy (Aggregation)



Part-of Hierarchy (Aggregation)



Is-Kind-of Hierarchy (Taxonomy)



Where are we?

- Three ways to deal with complexity:
  - Abstraction, Decomposition, Hierarchy
- Object-oriented decomposition is good
  - Unfortunately, depending on the purpose of the system, different objects can be found
- How can we do it right?
  - Start with a description of the functionality of a system
  - Then proceed to a description of its structure
- Ordering of development activities
  - Software lifecycle.

Models are not the Truth

- Karl Popper ("Objective Knowledge):
  - There is no absolute truth when trying to understand reality
  - One can only build theories, that are "true" until somebody finds a counter example
- Falsification: The act of disproving a theory or hypothesis
- The truth of a theory is never certain. We must use phrases like:
  - "by our best judgement", "using state-of-the-art knowledge"
- In software engineering any model is a theory:
  - We build models and try to find counter examples by:
    - Requirements validation, user interface testing, review of the design, source code testing, system testing, etc.
- Testing: The act of disproving a model.

Concepts and Phenomena

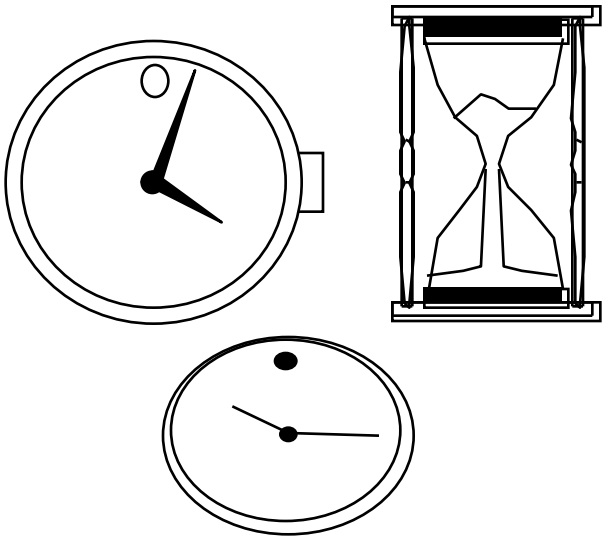
- Phenomenon
  - An object in the world of a domain as you perceive it
    - Examples: This lecture at 13:25, my black watch
- Concept
  - Describes the common properties of phenomena
    - Example: All lectures on software engineering
    - Example: All black watches
- A Concept is a 3-tuple:
  - **Name:** The name distinguishes the concept from other concepts
  - **Purpose:** Properties that determine if a phenomenon is a member of a concept
  - **Members:** The set of phenomena which are part of the concept.

Concepts, Phenomena, Abstraction and Modeling

Name **Purpose** **Members**

Watch

A device that measures time.



Definition Abstraction

- Classification of phenomena into concepts

Definition Modeling

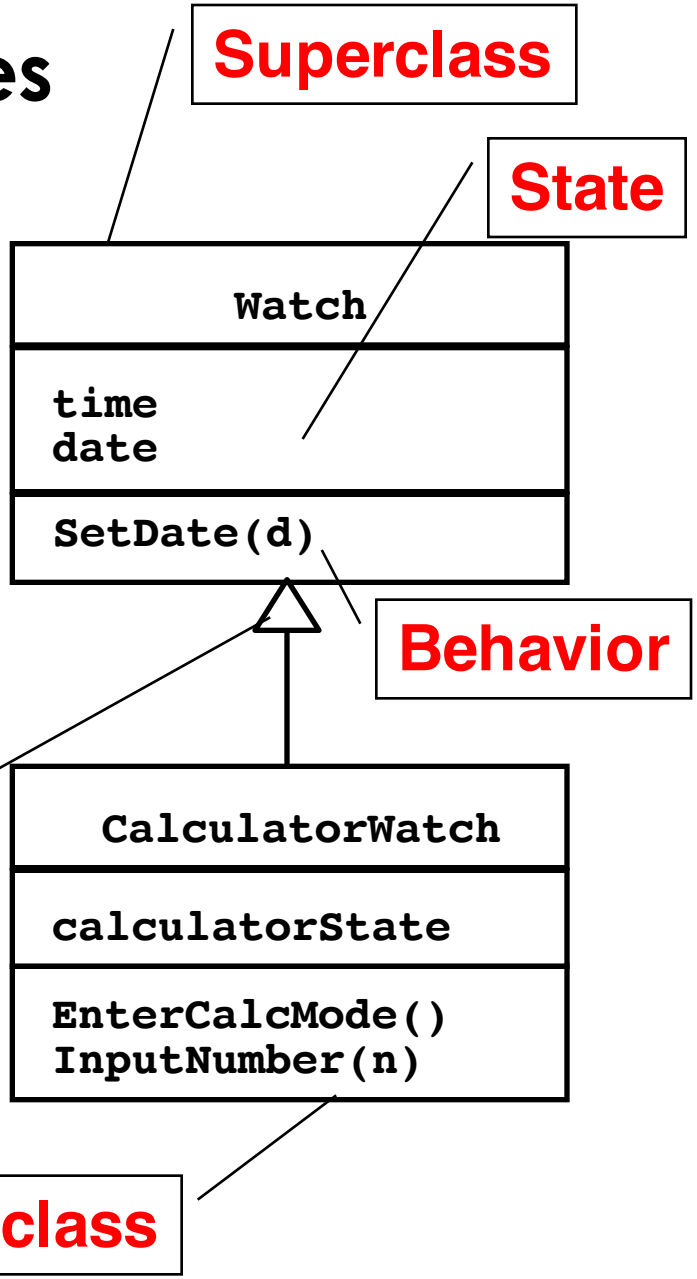
- Development of abstractions to answer specific questions about a set of phenomena while ignoring irrelevant details.

Abstract Data Types & Classes

- **Abstract data type**
  - A type whose implementation is hidden from the rest of the system
- **Class:**
  - An abstraction in the context of object-oriented languages
  - A class encapsulates state and behavior
    - Example: Watch

Unlike abstract data types, subclasses can be defined in terms of other classes using inheritance

- Example: CalculatorWatch



Type and Instance

- Type:
  - A concept in the context of programming languages
  - Name: int
  - Purpose: integral number
  - Members: `0,-1,1,2,-2,…`
  - Name: boolean
  - Purpose: logical
  - Members: {true, false}
- Instance:
  - Member of a specific type
- The type of a variable represents all possible instances of the variable

The following relationships are similar:

Type <-> Variable
Concept <-> Phenomenon
Class <-> Object

Systems

- A *system* is an organized set of communicating parts
  - Natural system: A system whose ultimate purpose is not known
  - Engineered system: A system which is designed and built by engineers for a specific purpose
- The parts of the system can be considered as systems again
  - In this case we call them *subsystems*

Examples of natural systems:

- Universe, earth, ocean

Examples of engineered systems:

- Airplane, watch, GPS

Examples of subsystems:

- Jet engine, battery, satellite.

Systems, Models and Views

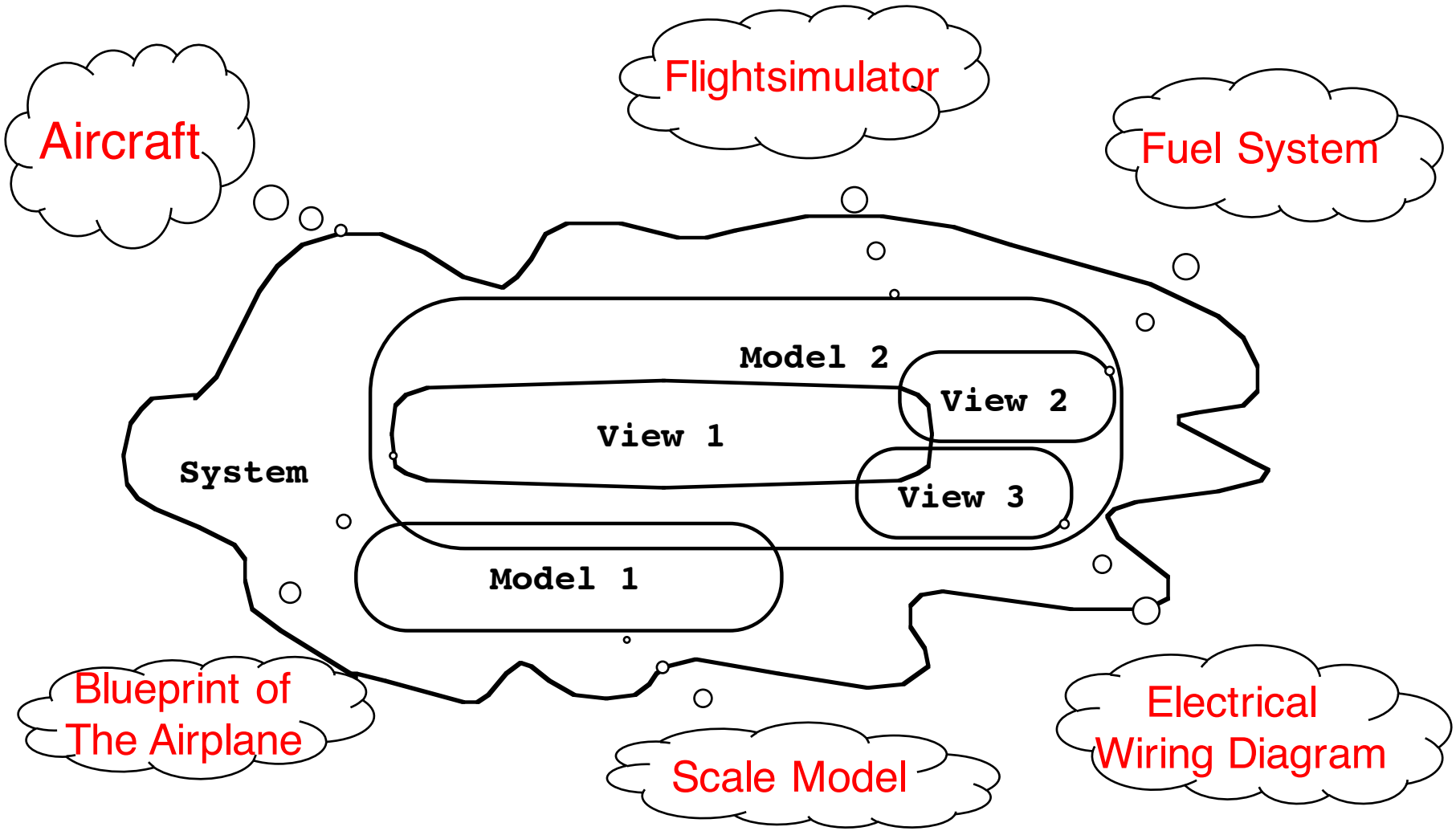
- A ***model*** is an abstraction describing a system
- A ***view*** depicts selected aspects of a model
- A ***notation*** is a set of graphical or textual rules for depicting models and views:
  - Informal notations ("napkin design")
  - Formal notations (UML)

System:
Airplane

Model:
Flight simulator
Scale model

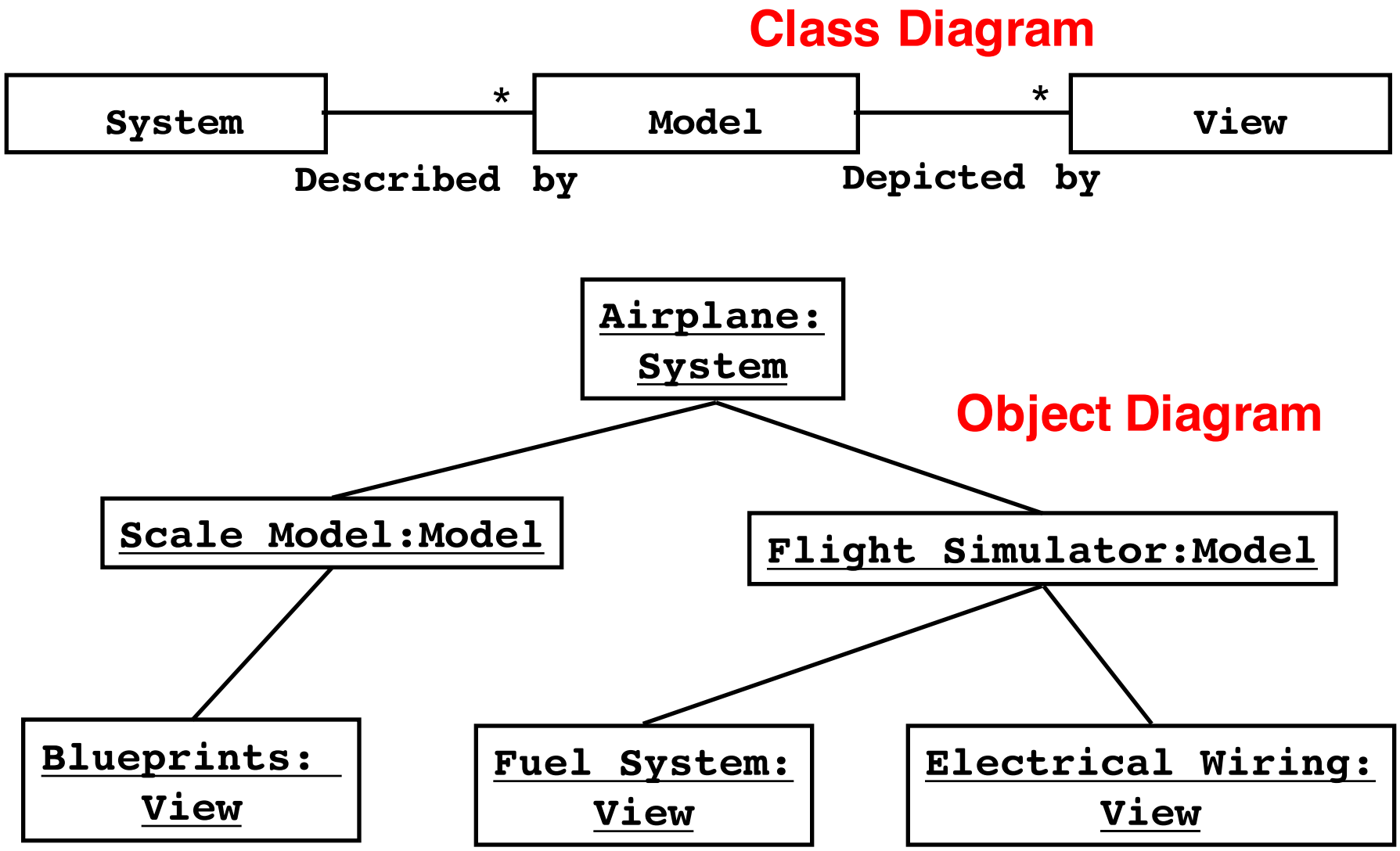
View:
Blueprint of the airplane
Electrical wiring diagram
An airplane breaking the sound barrier

Systems, Models and Views ("Napkin" Notation)



Views and models of a complex system usually overlap

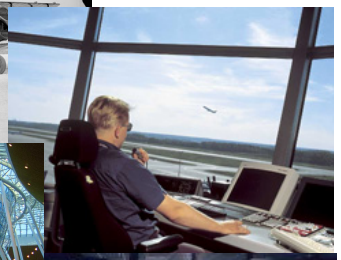
Systems, Models and Views (UML Notation)

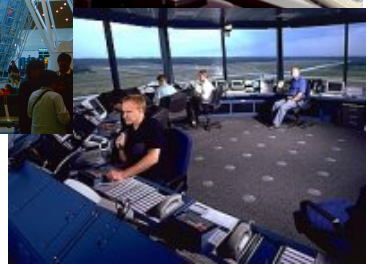


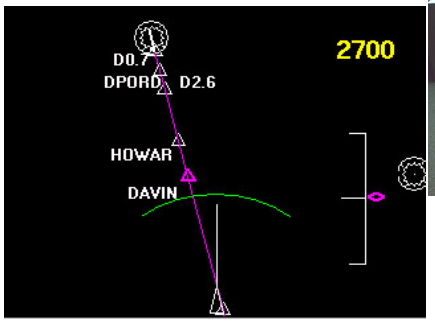
UML

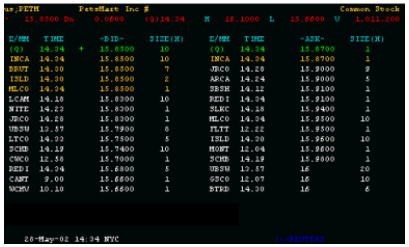
- UML (Unified Modeling Language)
  - A standard for modeling software systems
  - Object Management Group: http://www.uml.org/
  - Convergence of notations used in object-oriented methods
    - OMT (James Rumbaugh and colleagues)
    - Booch (Grady Booch)
    - OOSE (Ivar Jacobson)
- Current Version: UML 2.5 (July 2015)
  - Basic Information at http://www.uml.org/what-is-uml.htm
- Commercial tools: Visual Paradigm, Rational (IBM),Together (Borland), Visual Architect (business processes, BCD)
- Open Source tools: ArgoUML, StarUML, Umbrello, Unicase
- Commercial and Opensource: PoseidonUML (Gentleware)

Object-oriented Modeling









Application Domain
(Phenomena)

Solution Domain
(Phenomena)

System Model (Concepts) *Analysis*

System Model (Concepts) *Design*

UML Package

TrafficControl

Aircraft

TrafficController

Airport

FlightPlan

MapDisplay

Textual Display

FlightPlanDatabase

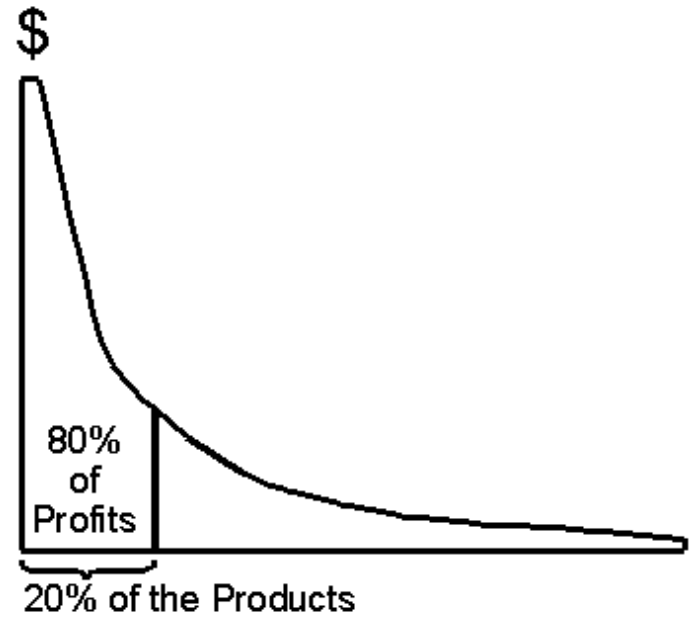
TrafficControl

Application vs Solution Domain

- Application Domain (Analysis):
  - The environment in which the system is operating
- Solution Domain (Design, Implementation):
  - The technologies used to build the system
- Both domains contain abstractions that we can use for the construction of the system model.

UML: First Pass

- You can solve 80% of the modeling problems by using 20 % UML
- We teach you those 20%
- 80-20 rule: Pareto principle



Vilfredo Pareto, 1848-1923
Introduced the concept of Pareto Efficiency,
Founder of the field of microeconomics.

UML First Pass

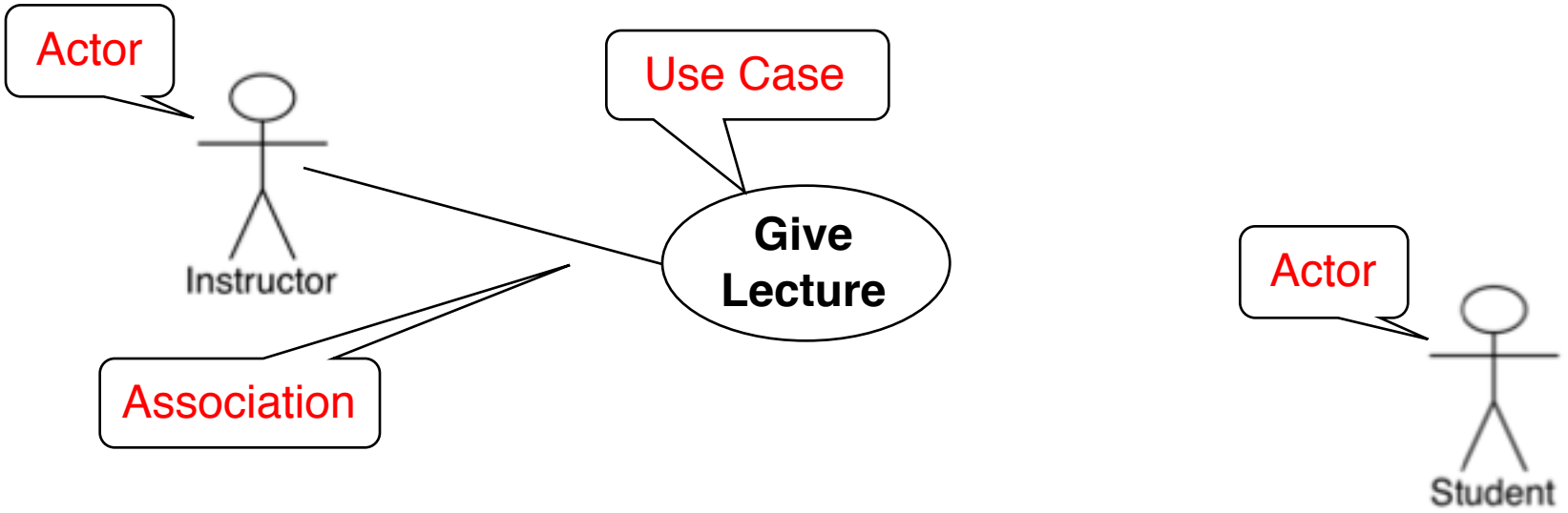
- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams, also called State machine diagrams
  - Describe the dynamic behavior of a single, individual object.

UML Diagrams: Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities in the application or solution domain
    - Rectangles denote classes, instances, system boundaries
    - Ovals denote functions
    - UML also allows graphical icons for nodes: 
  - Names of classes are not underlined
    - `SimpleWatch`
    - `Firefighter`
  - Names of instances are underlined
    - `myWatch:SimpleWatch`
    - `Joe:Firefighter`
  - An edge between two nodes denotes a relationship between the corresponding entities
    - Edges are called associations in UML diagrams.

UML first pass: Use Case Diagrams

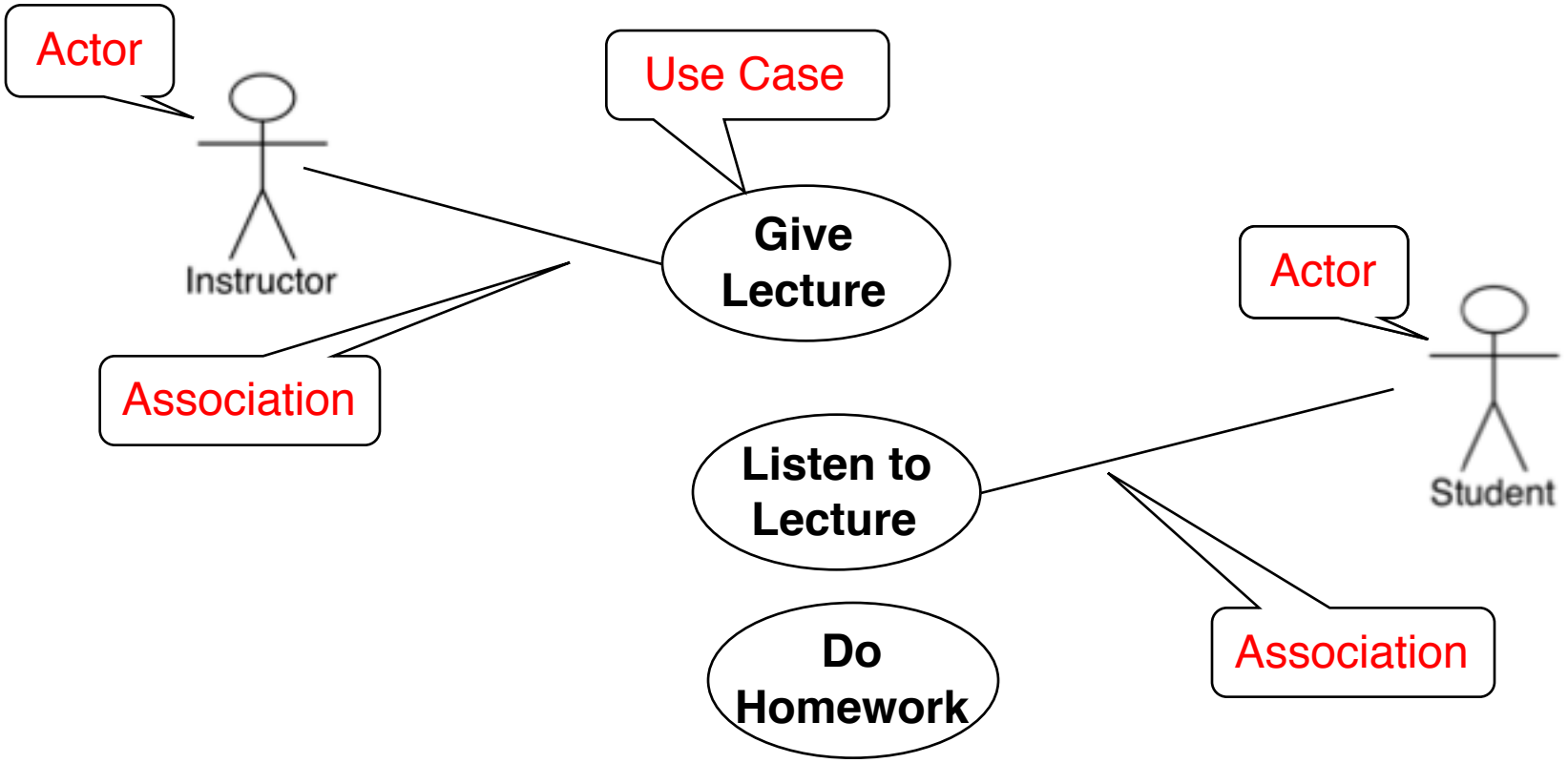
Example: Modeling a Lecture



Use case diagrams represent the functionality of a system from user's point of view

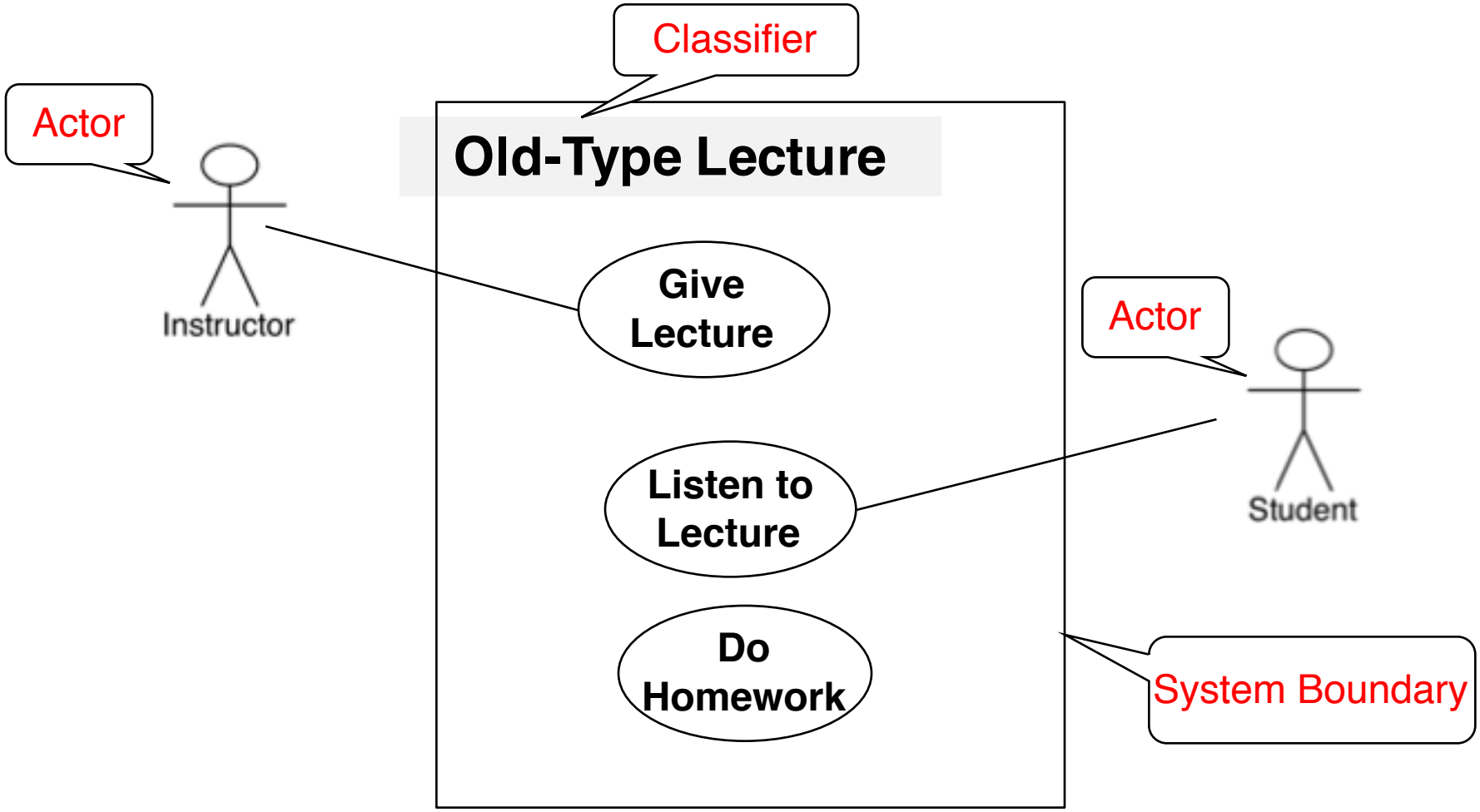
UML first pass: Use Case Diagrams

Example: Modeling a Lecture



Use case diagrams represent the functionality of a system from user's point of view

UML first pass: Use Case Diagrams

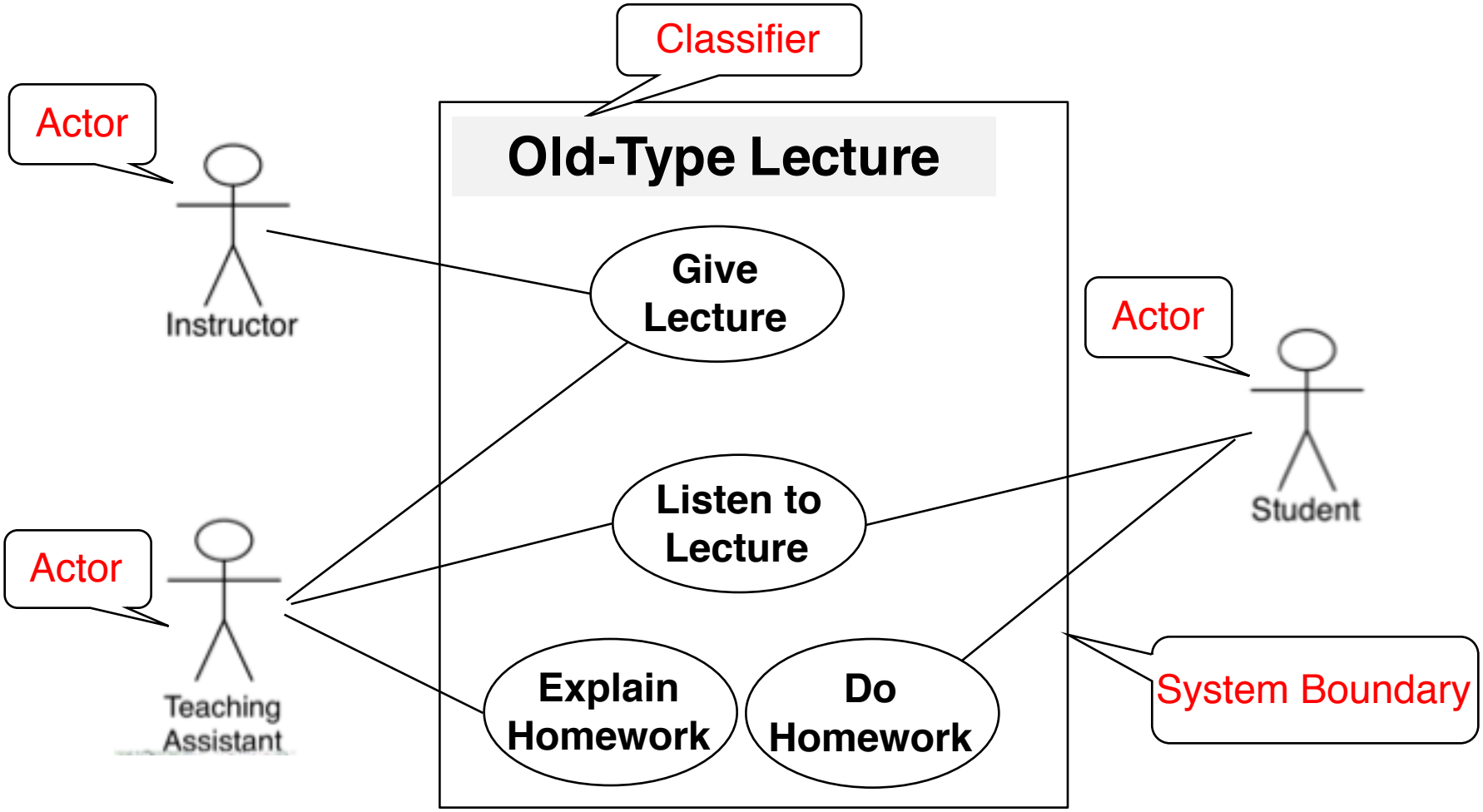


Use case diagrams represent the functionality of a system from user's point of view

Where is the System Boundary here?

What about the Actor Pedestrian?

UML first pass: Use Case Diagrams



Use case diagrams represent the functionality of a system from user's point of view

In-Class Exercise #1

- Problem Statement (in Natural Language):

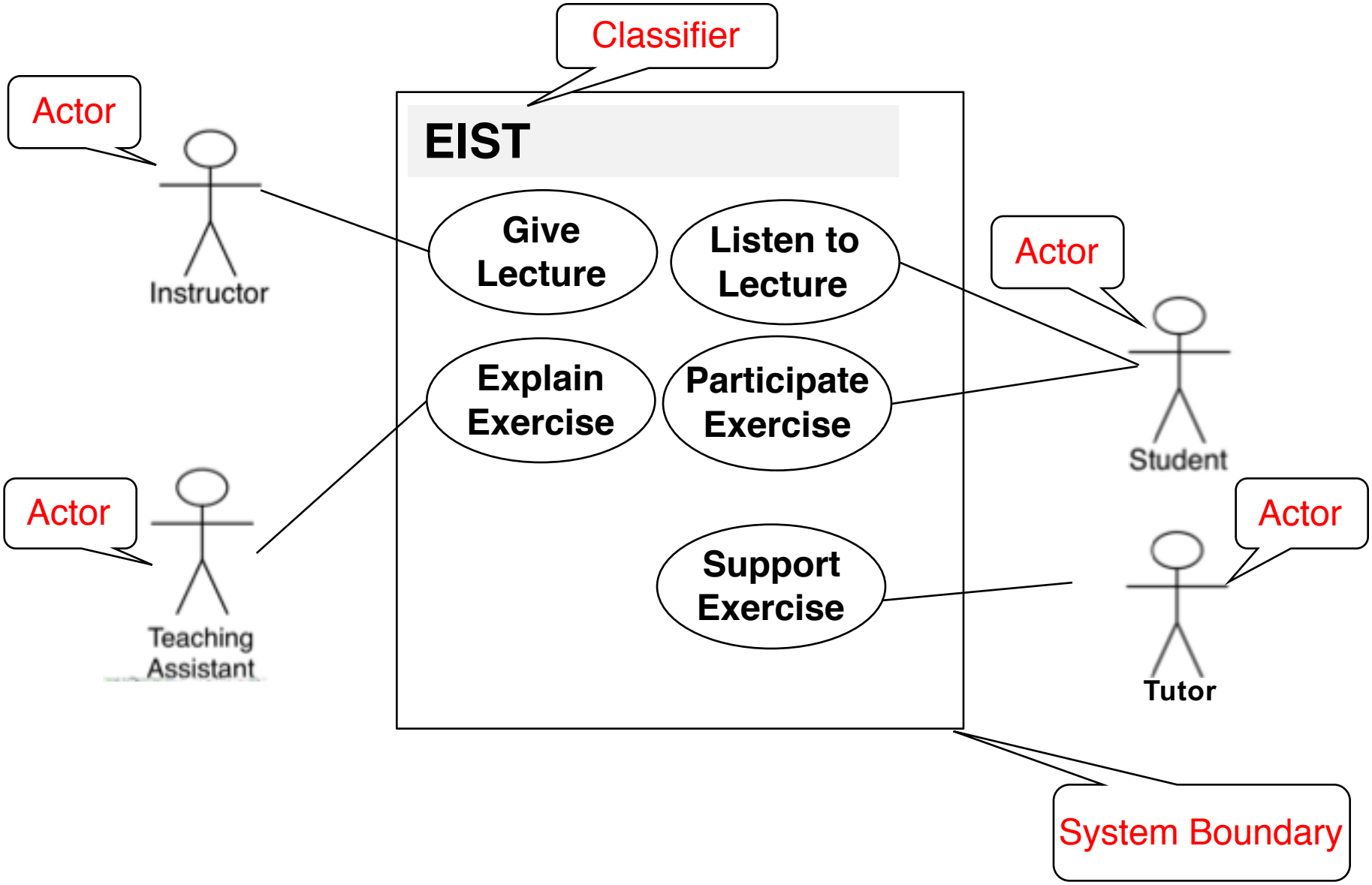
  During the EIST course an instructor gives lectures to students. Students are listening to the lecture and also participate in exercises during the lecture. Exercises are explained by a teaching assistant and are supported by several tutors in order to help students participating in the exercise.

- Your Task:
  - Model this statement in UML

Details for the Submission

1. Draw an informal UML model
2. Take a picture of it
3. Upload the picture to Moodle using the Exercise #1 Model EIST upload form
4. Cross fingers and pray ;-)
5. The first 3 winners will get a gummy bear
6. Among all the other submissions we will do a random drawing to determine 3 additional winners
7. Estimated exercise time is 10 minutes. Moodle will close at the end of the class.

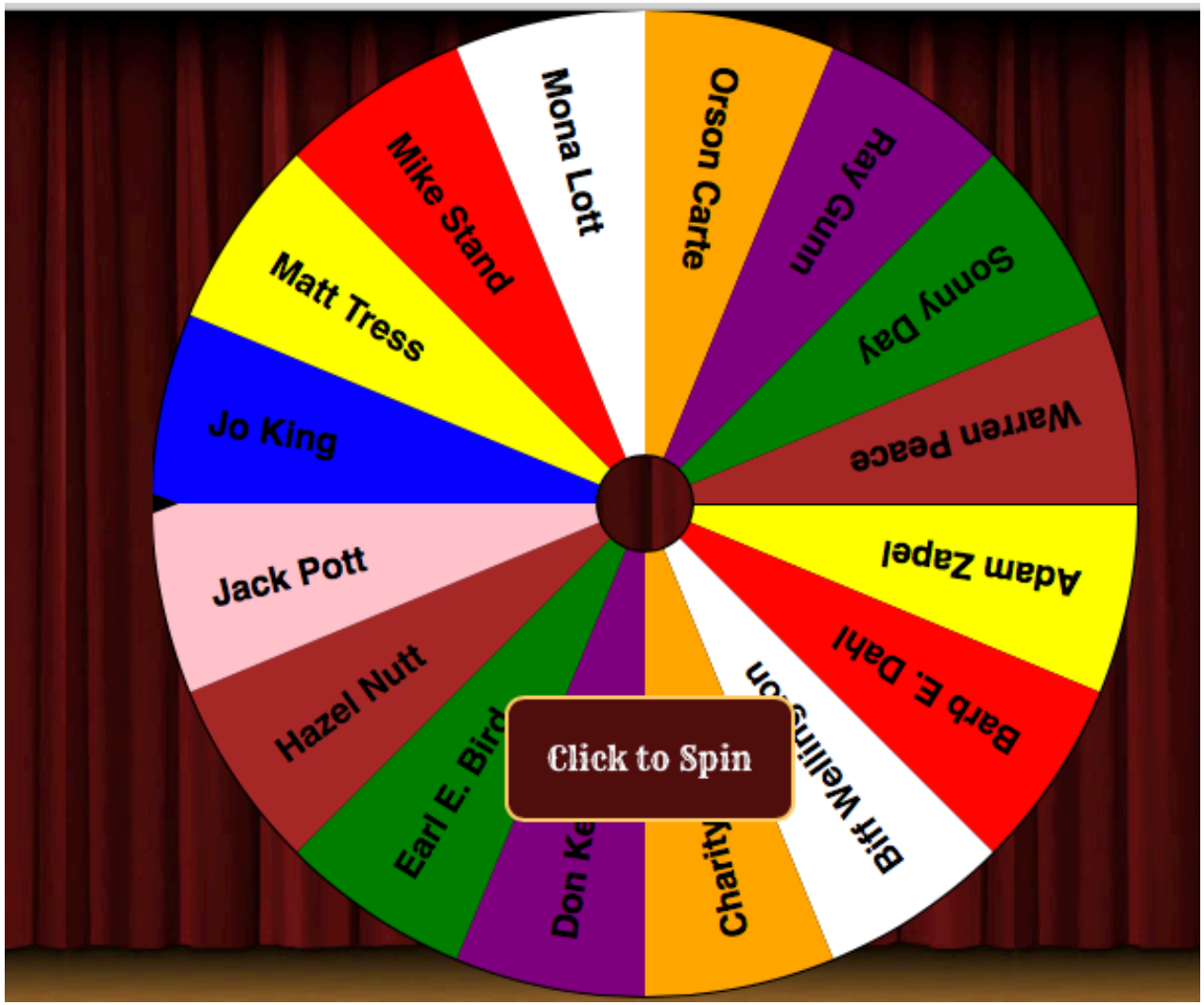
Solution #1: Informal UML Model of EIST



Exercise #1 Winners

- Rieger, Maximilian
- Gruner, Fabian
- Nöller, Hendik

Determination of the Additional Winners



UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- Today we concentrated on the use case notation:
  - Functional model: Use case diagram
- Next lecture (Thursday, 21. April 2016)
- Modeling UML, Part 2
  - Object model
    - Class diagram
  - Dynamic model

References

Bernd Bruegge, Allen H. Dutoit

- Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson Education, 2009

- Martin Fowler
  - UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed., Addison-Wesley, 2003
- Grady Booch,James Rumbaugh,Ivar Jacobson
  - The Unified Modeling Language User Guide, Addison Wesley, 2nd edition, 2005.