

Modeling with UML: Basic Notations

**Software Engineering I
Lecture 2
18 April 2007**

Prof. Bernd Bruegge, Ph.D.
Applied Software Engineering
Technische Universitaet Muenchen

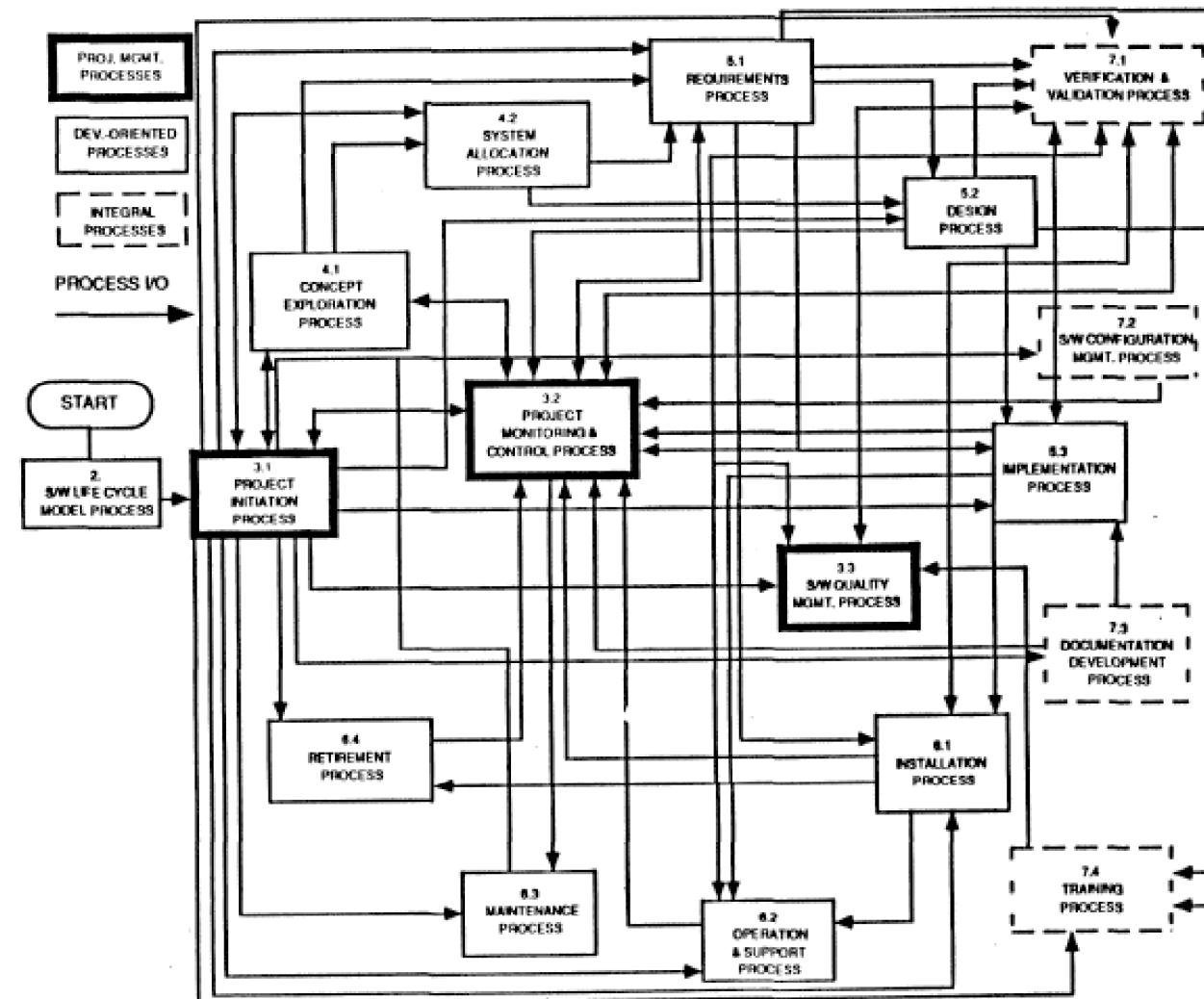
Odds and Ends (1)

- Registration for the Exercises
 - Started yesterday
 - Any problems?
- Deadline for registration
 - Friday, April 20 at 12:00
- First group meeting:
 - Monday, April 23 at 10:00

Odds and Ends (2)

- **Reading for this Week:**
 - Chapter 1 and 2, Bruegge&Dutoit, Object-Oriented Software Engineering
- **Software Engineering I Portal**
 - <http://wwwbruegge.in.tum.de/twiki/bin/view/Lehrstuhl/SoftwareEngineeringSoSe2007>
- **Lectures Slides:**
 - Will be posted after each lecture.

Overview for the Lecture



Overview for the Lecture

- Three ways to deal with complexity
 - Abstraction and Modeling
 - Decomposition
 - Hierarchy
- Introduction into the UML notation
- First pass on:
 - Use case diagrams
 - Class diagrams
 - Sequence diagrams
 - Statechart diagrams
 - Activity diagrams

Abstraction

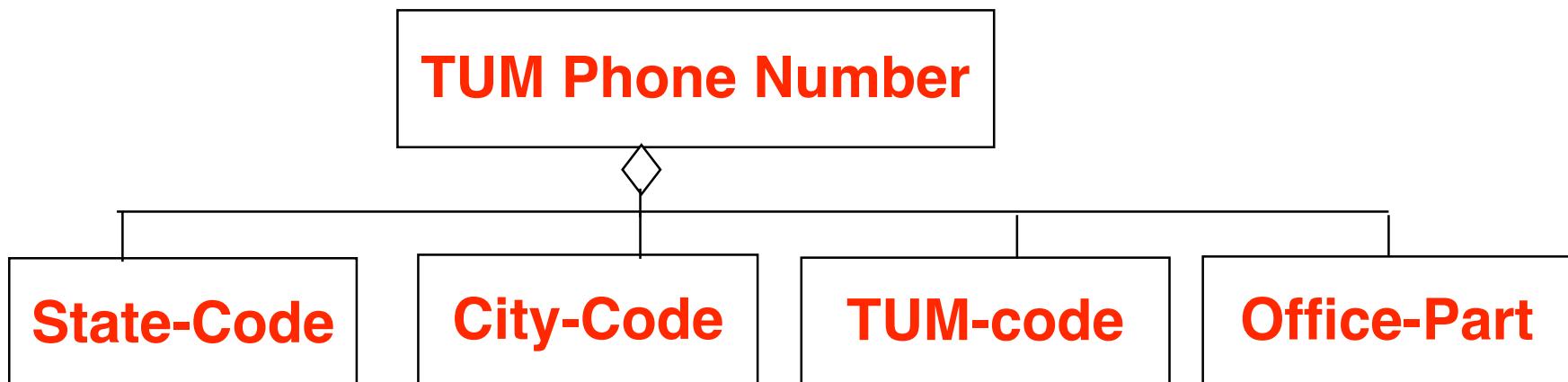
- Complex systems are hard to understand
 - The 7 +- 2 phenomena
 - Our short term memory cannot store more than 7+-2 pieces at the same time -> **limitation of the brain**
 - TUM Phone Number: 498928918204

Abstraction

- Complex systems are hard to understand
 - The 7 +- 2 phenomena
 - Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
 - TUM Phone Number: 498928918204
- Chunking:
 - Group collection of objects to reduce complexity
 - 4 chunks:
 - State-code, city-code, TUM-code, Office-Part

Abstraction

- Complex systems are hard to understand
 - The 7 +- 2 phenomena
 - Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
 - TUM Phone Number: 498928918204
- Chunking:
 - Group collection of objects to reduce complexity
 - State-code, city-code, TUM-code, Office-Part



Abstraction

- Abstraction allows us to ignore unessential details
- Two definitions for abstraction:
 - Abstraction is a *thought process* where ideas are distanced from objects
 - **Abstraction as activity**
 - Abstraction is the *resulting idea* of a thought process where an idea has been distanced from an object
 - **Abstraction as entity**
- Ideas can be expressed by models



Model

- A model is an abstraction of a system
 - A system that no longer exists
 - An existing system
 - A future system to be built.



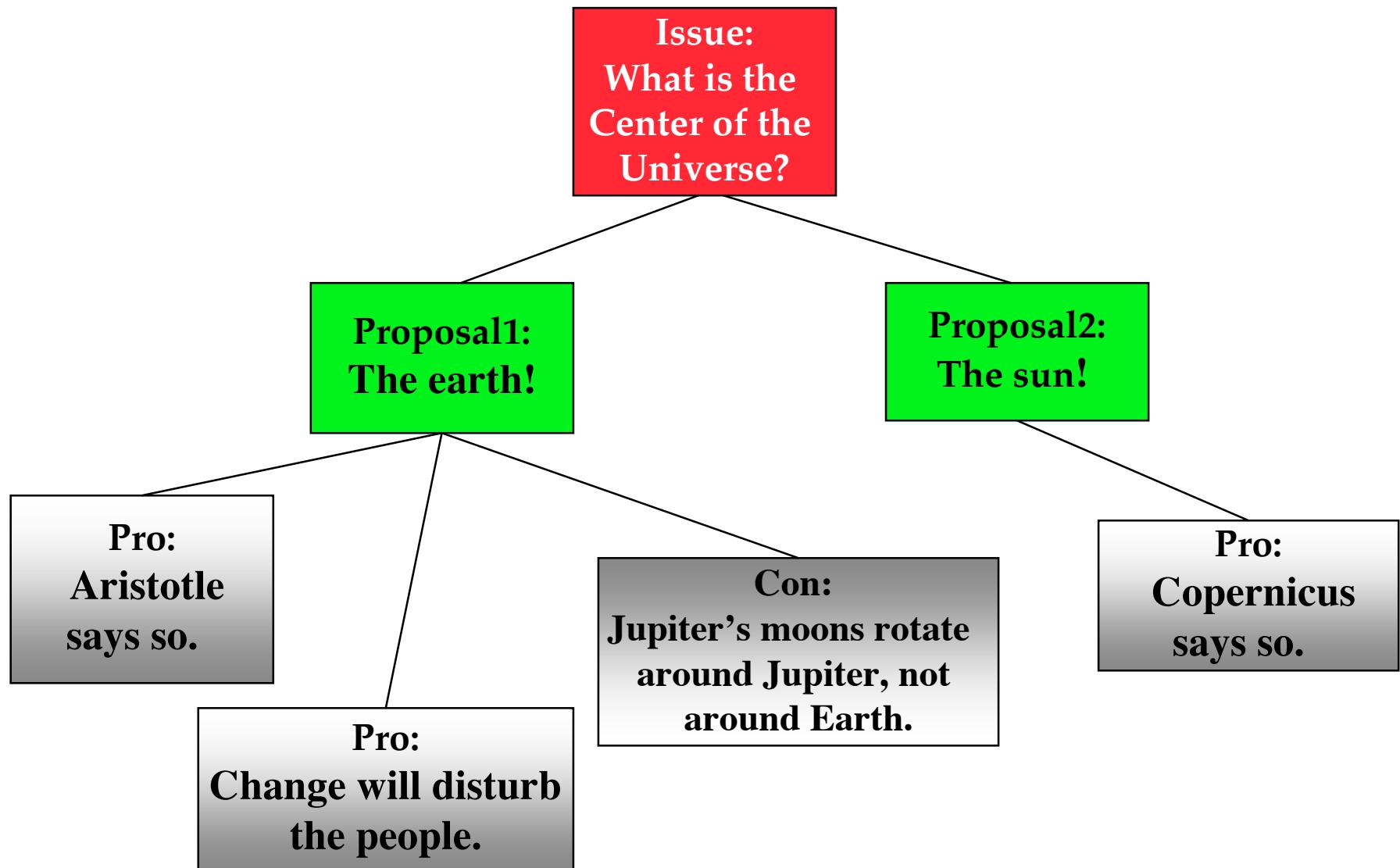
We use Models to describe Software Systems

- **Object model:** What is the structure of the system?
- **Functional model:** What are the functions of the system?
- **Dynamic model:** How does the system react to external events?
- **System Model:** Object model + functional model + dynamic model

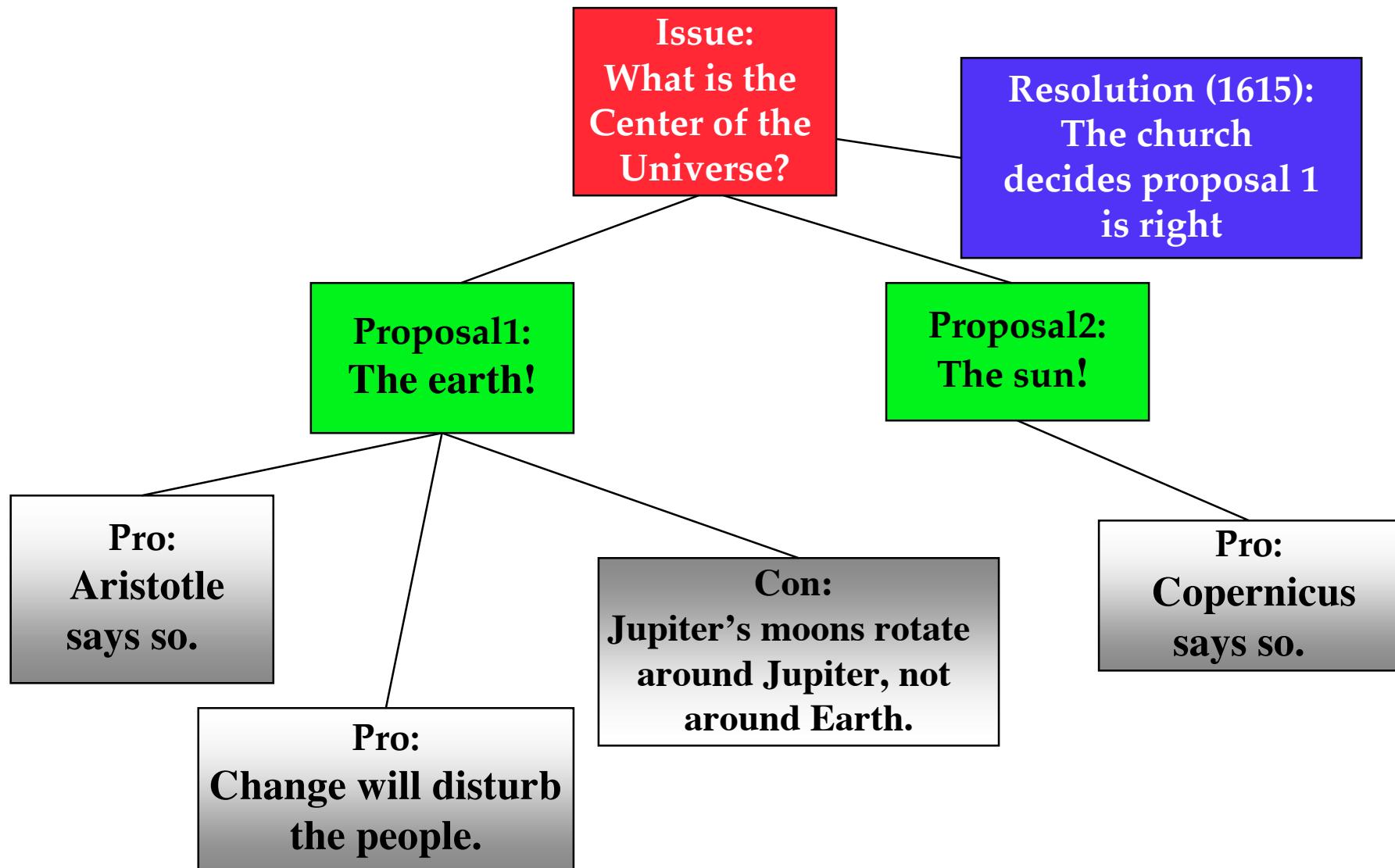
Other models used to describe Software System Development

- Task Model:
 - PERT Chart: What are the dependencies between tasks?
 - Schedule: How can this be done within the time limit?
 - Organization Chart: What are the roles in the project?
- Issues Model:
 - What are the open and closed issues?
 - What blocks me from continuing?
 - What constraints were imposed by the client?
 - What resolutions were made?
 - These lead to action items

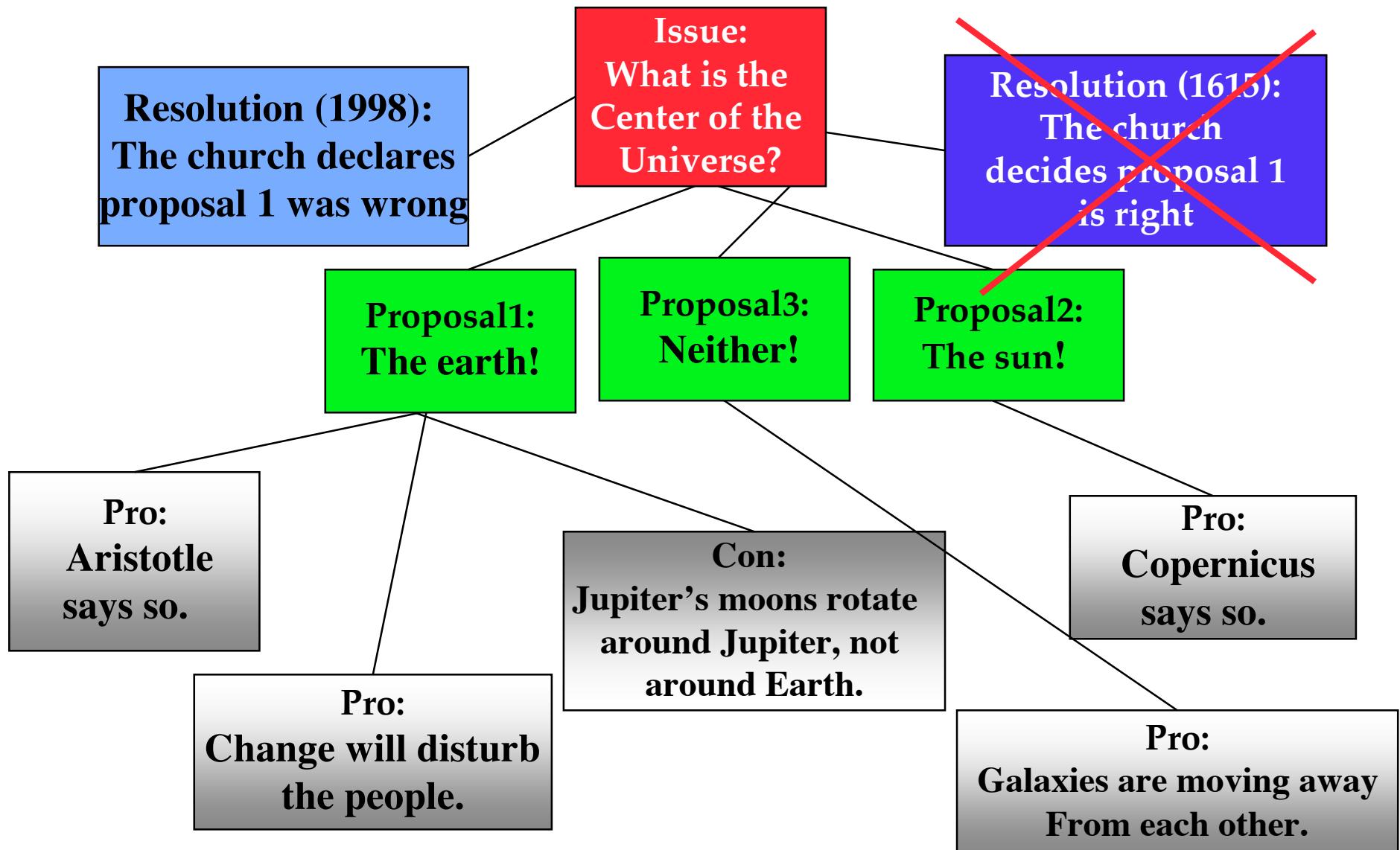
Issue-Modeling



Issue-Modeling



Issue-Modeling



2. Technique to deal with Complexity: Decomposition

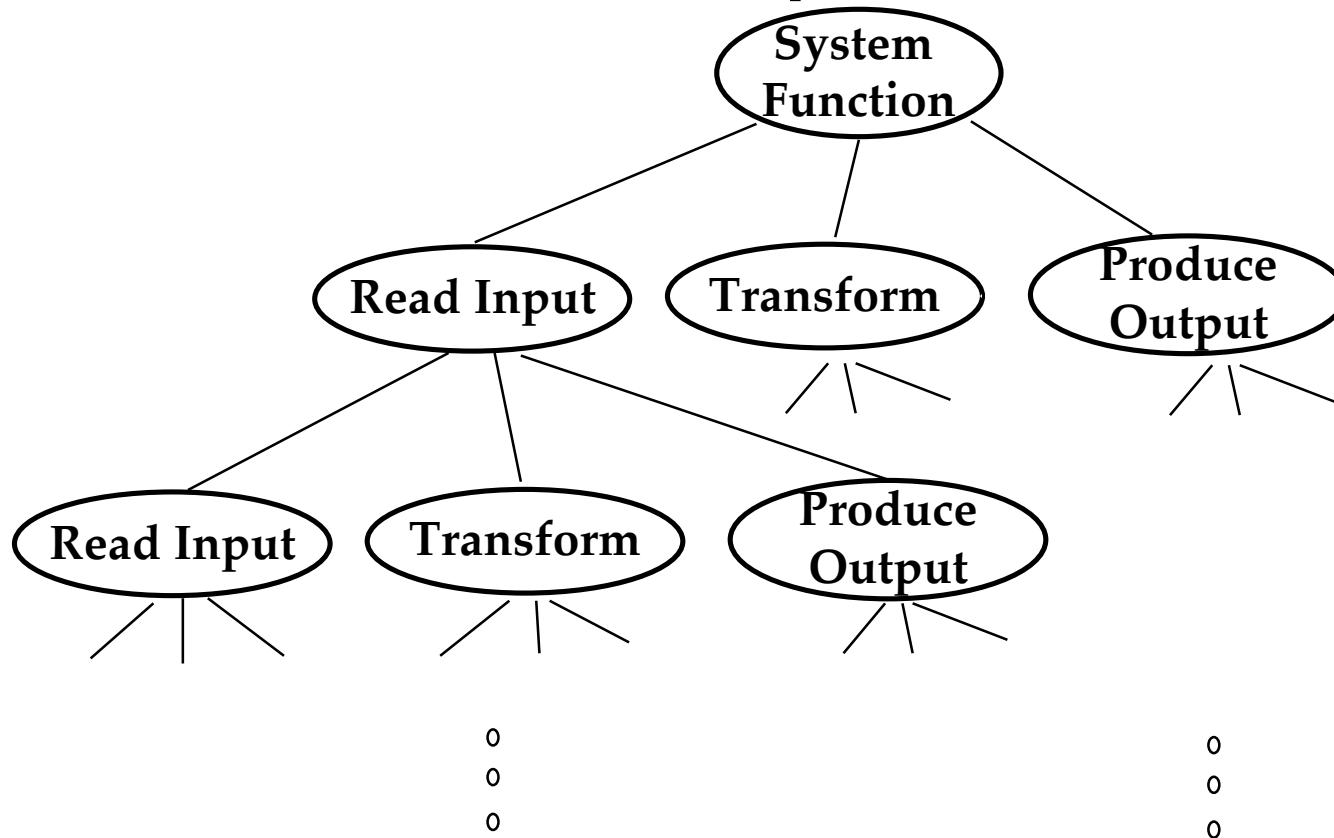
- A technique used to master complexity (“divide and conquer”)
- Two major types of decomposition
 - Functional decomposition
 - Object-oriented decomposition
- **Functional decomposition**
 - The system is decomposed into modules
 - Each module is a major function in the application domain
 - Modules can be decomposed into smaller modules.

Decomposition (cont'd)

- Object-oriented decomposition
 - The system is decomposed into classes ("objects")
 - Each class is a major entity in the application domain
 - Classes can be decomposed into smaller classes
- Object-oriented vs. functional decomposition

Which decomposition is the right one?

Functional Decomposition



Top Level functions

Level 1 functions

Level 2 functions



0 0 0 0



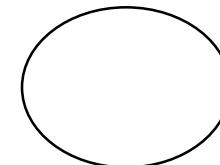
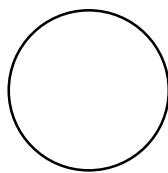
Machine instructions

Functional Decomposition

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

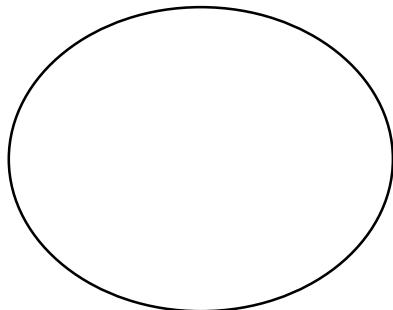
Functional Decomposition

- The functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
 - Source code is hard to understand
 - Source code is complex and impossible to maintain
 - User interface is often awkward and non-intuitive
- Example: Microsoft Powerpoint's Autoshapes
 - How do I change a square into a circle?

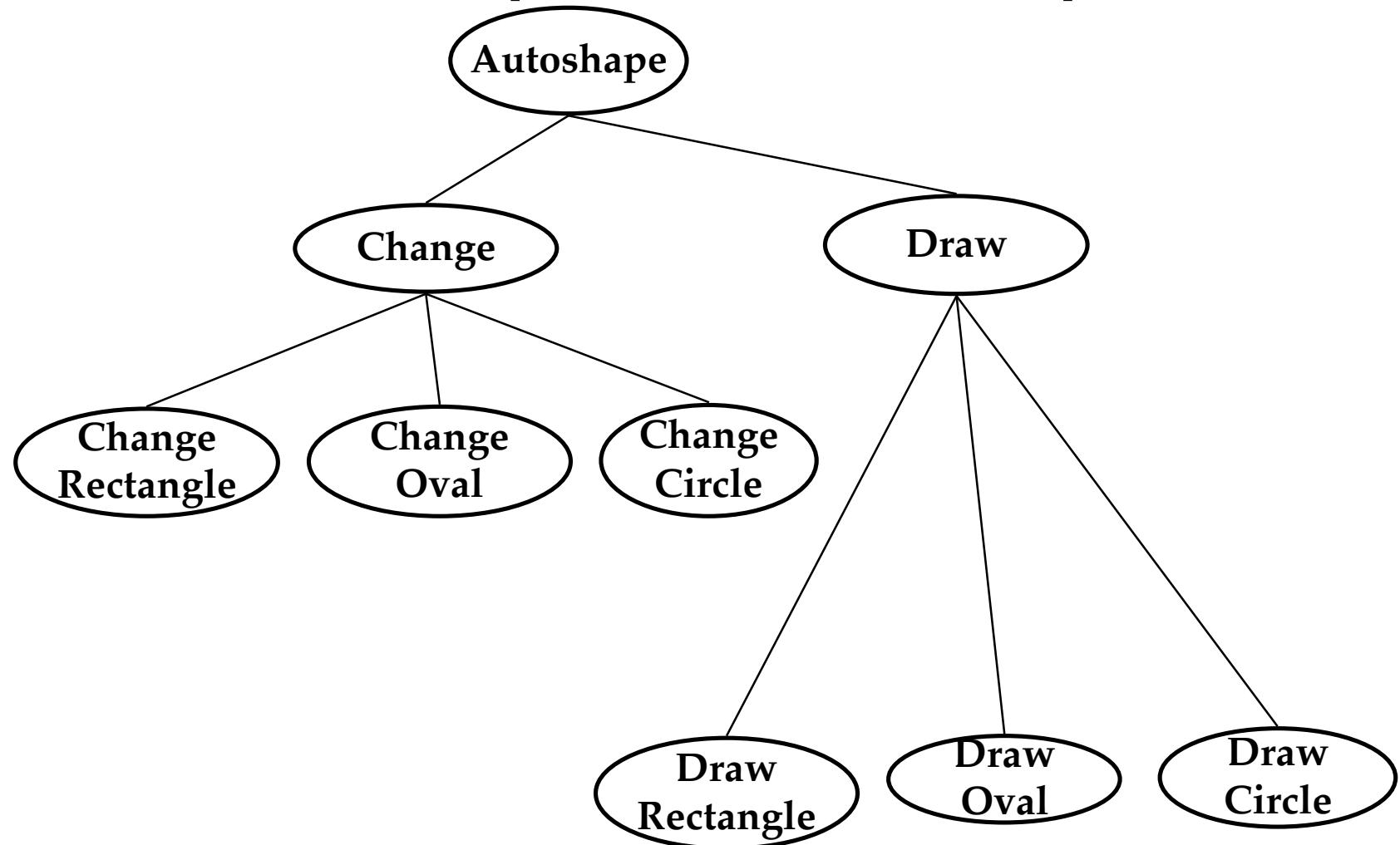


?

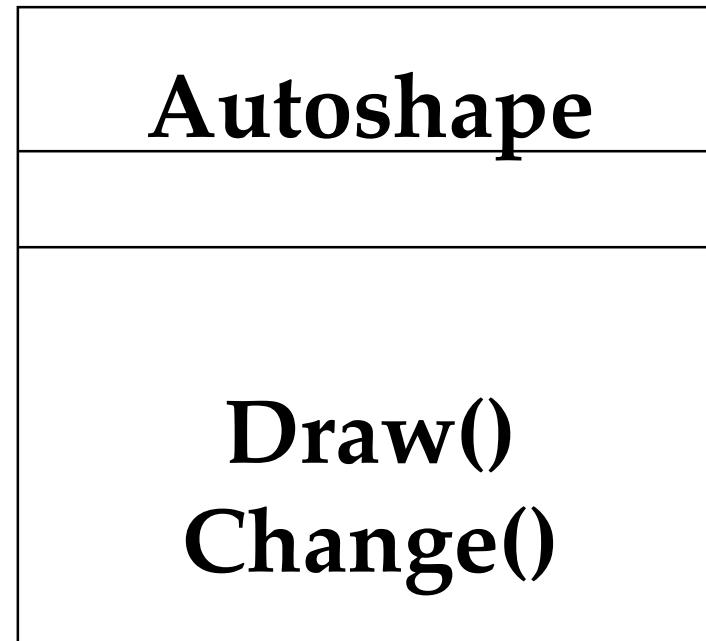
Changing a Square into a Circle



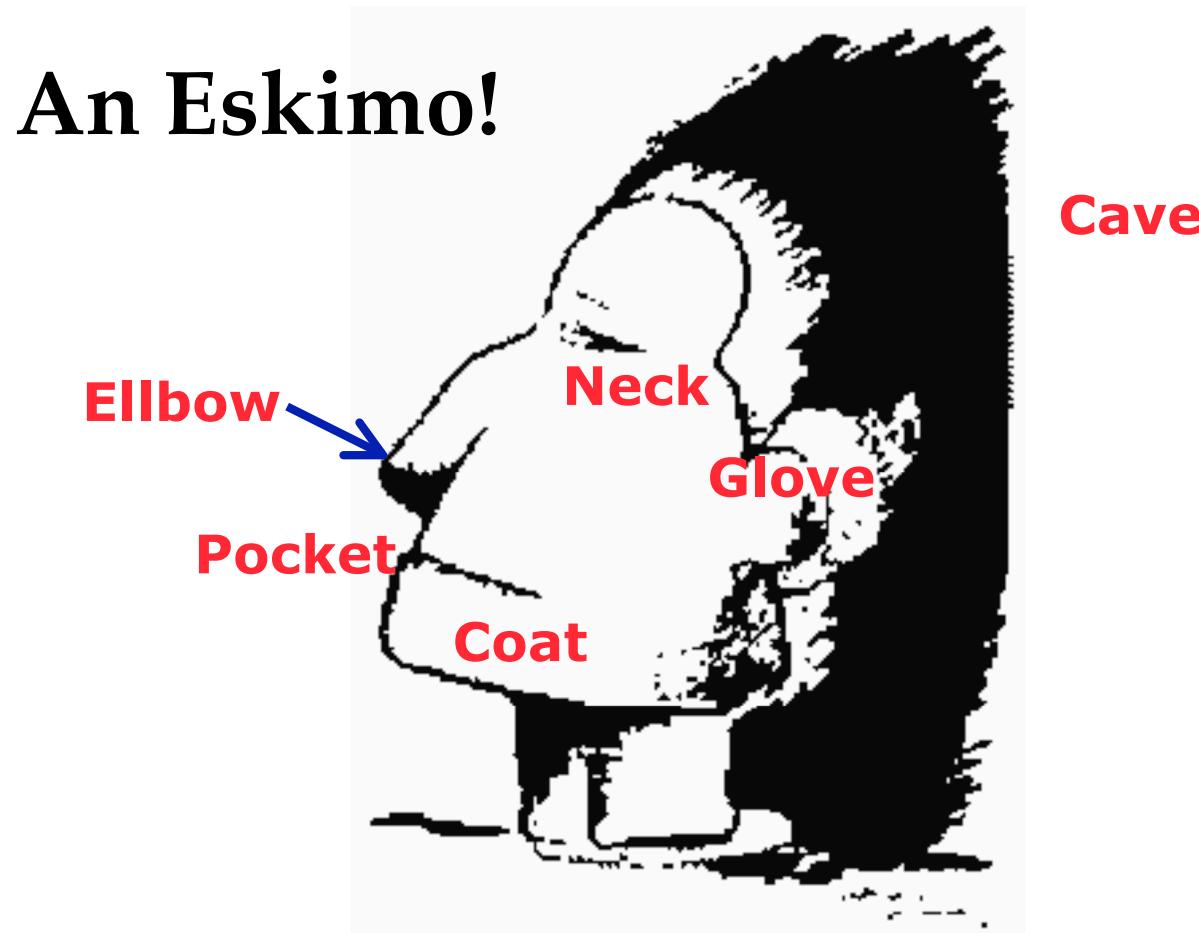
Functional Decomposition: Autoshape



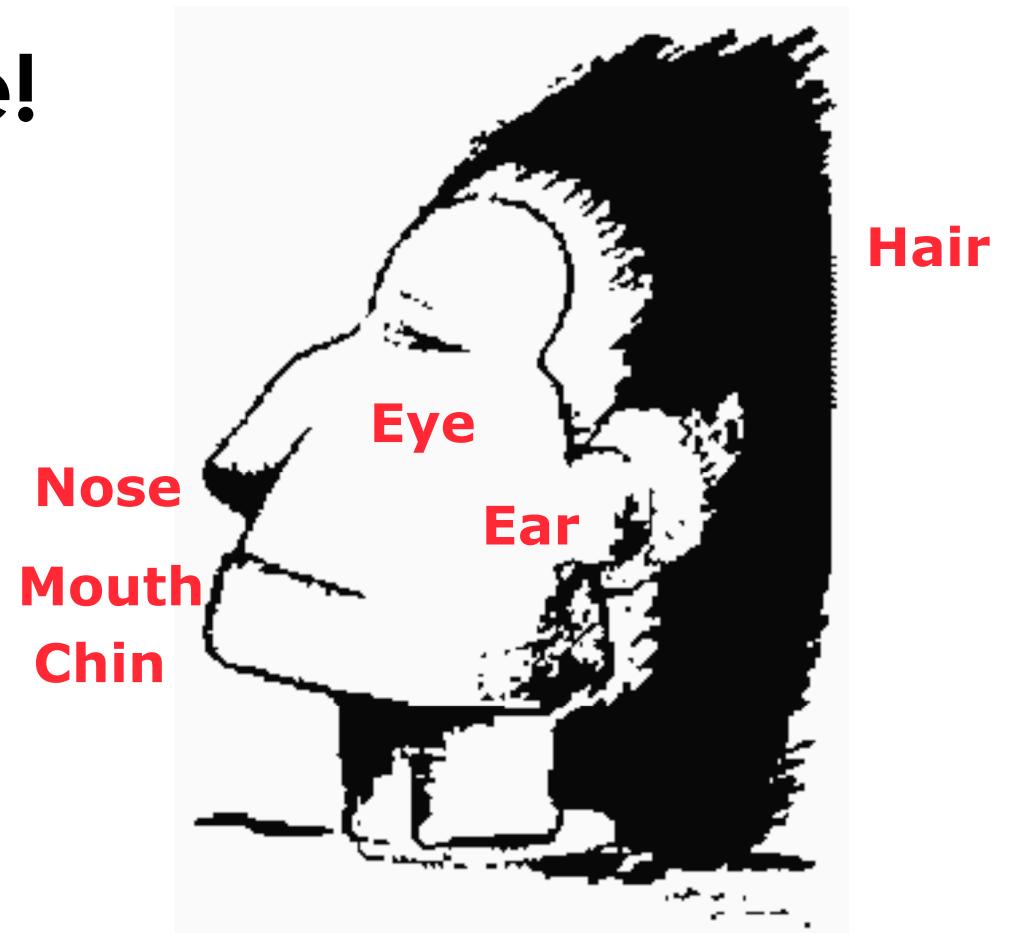
Object-Oriented View



What is This?



A Face!

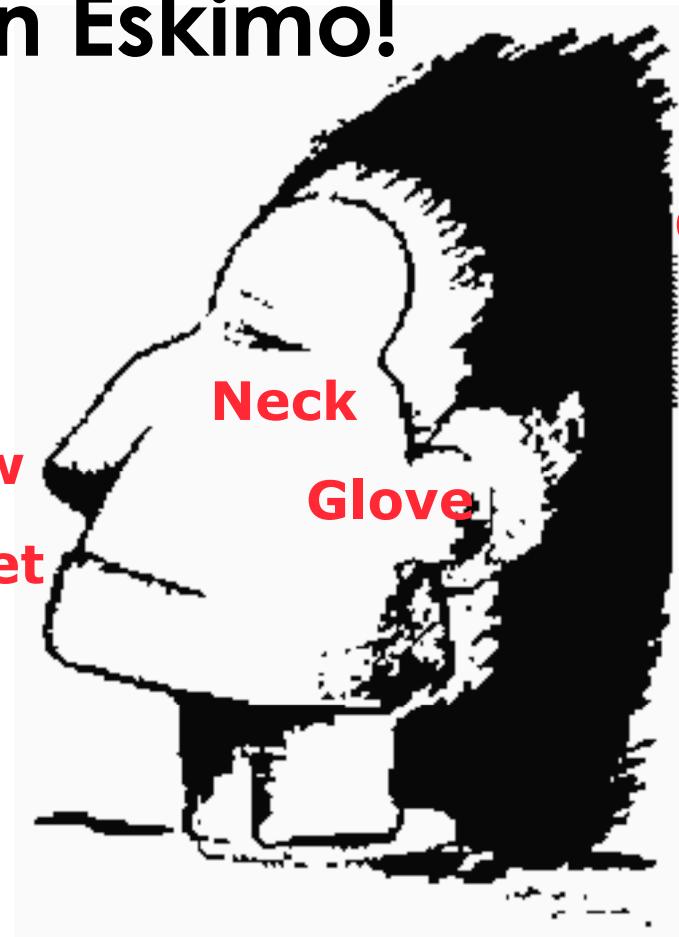


An Eskimo!

Elbow
Pocket
Coat

Neck
Glove

Cave

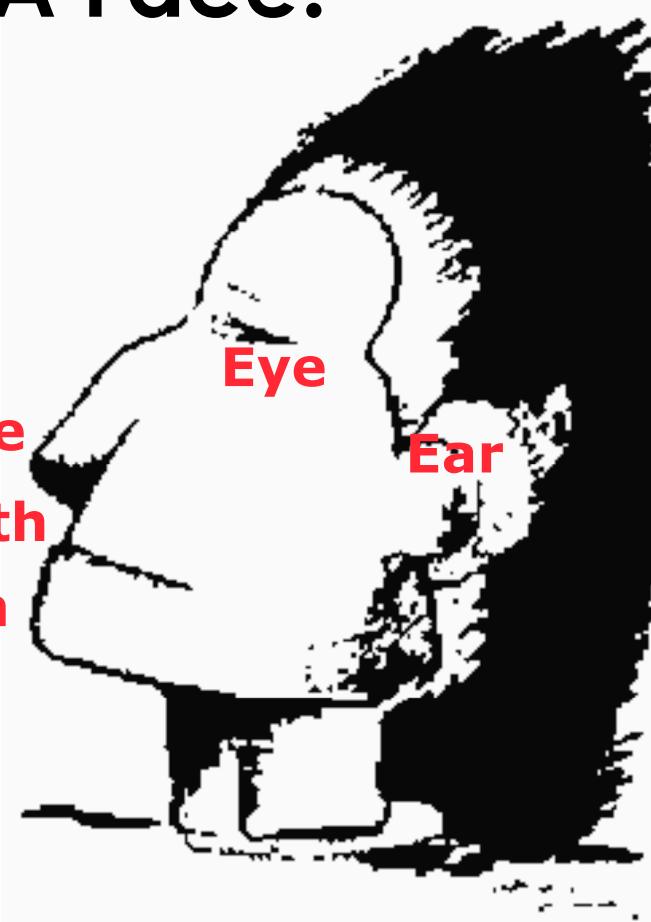


A Face!

Nose
Mouth
Chin

Eye
Ear

Hair



Class Identification

- **Basic assumptions:**
 - We can find the *classes for a new software system*: **Greenfield Engineering**
 - We can identify the *classes in an existing system*: **Reengineering**
 - We can create a *class-based interface to an existing system*: **Interface Engineering**

Class Identification (cont'd)

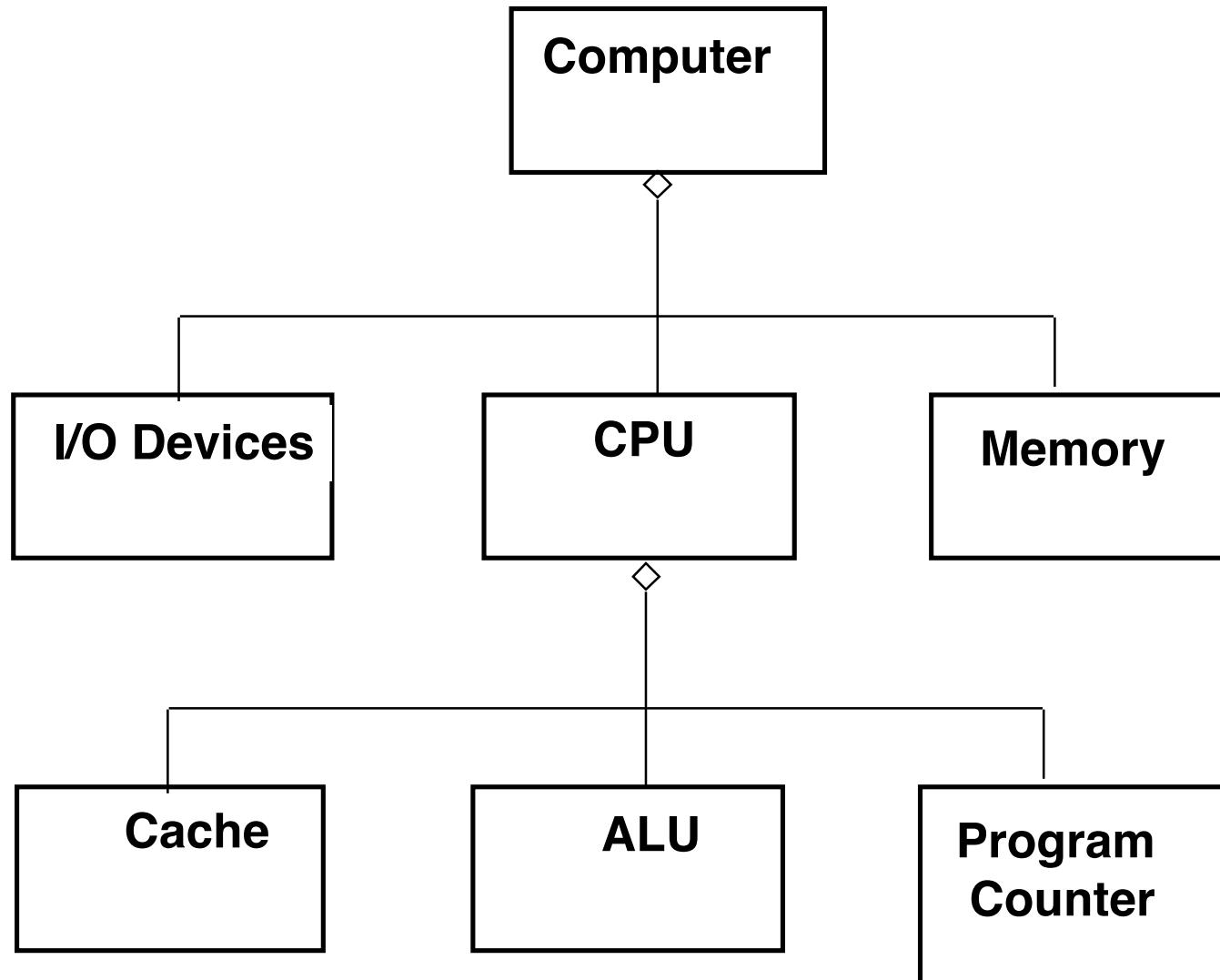
- **Why can we do this?**
 - Philosophy, science, experimental evidence
- **What are the limitations?**
 - Depending on the purpose of the system, different objects might be found
- **Crucial**

Identify the purpose of a system

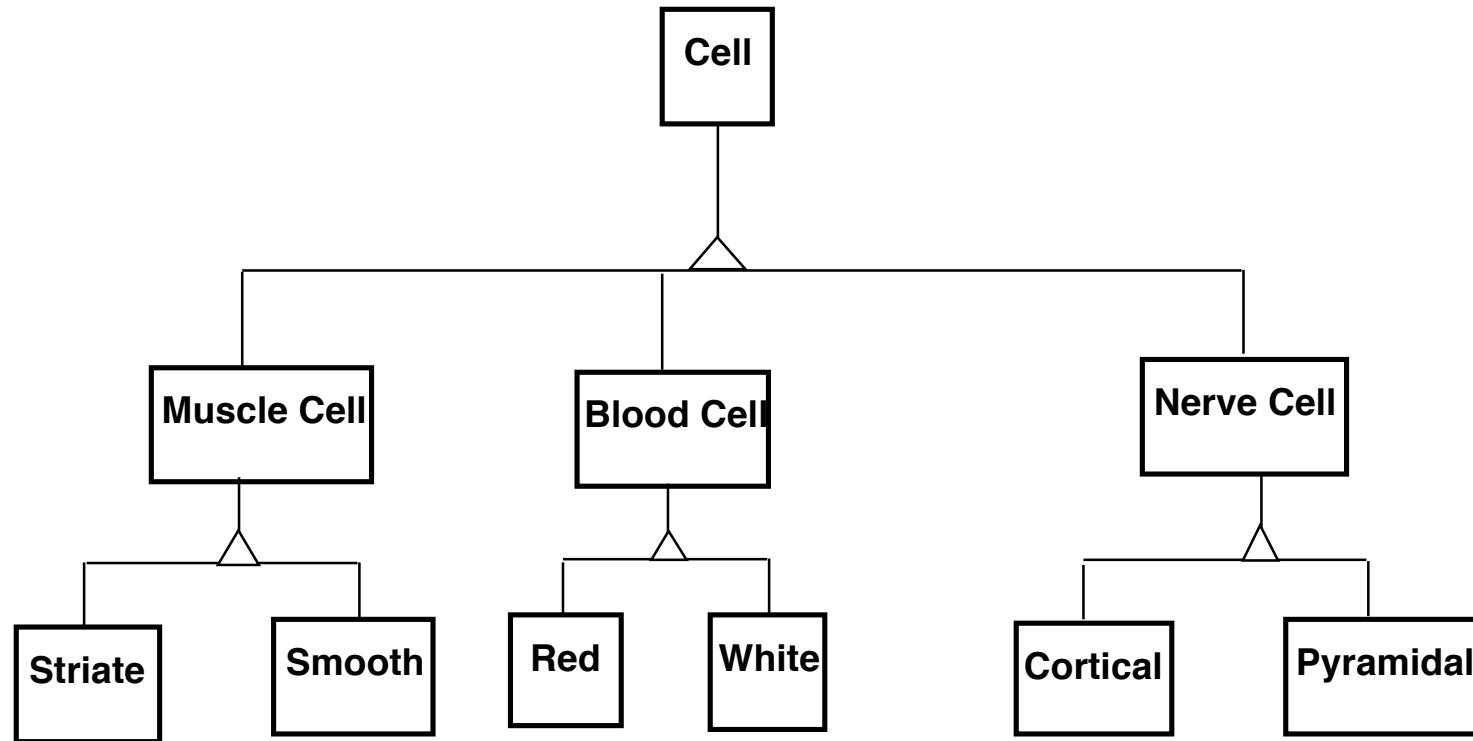
3. Hierarchy

- So far we got abstractions
 - This leads us to classes and objects
 - “Chunks”
- 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

Part-of Hierarchy (Aggregation)



Is-Kind-of Hierarchy (Taxonomy)



Where are we now?

- 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

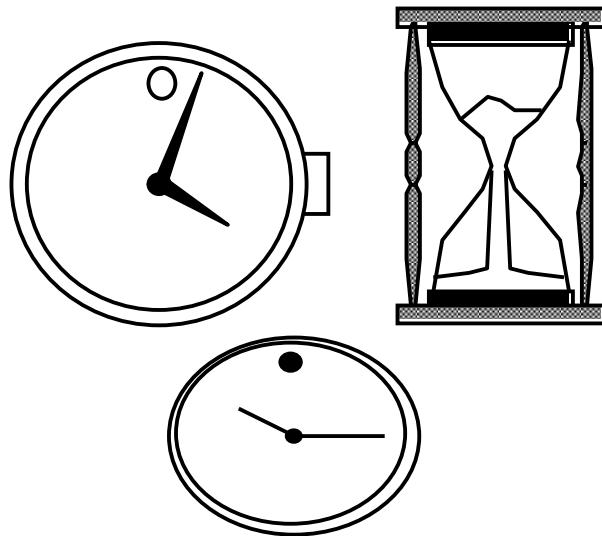
Models must be falsifiable

- 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 on April 18 at 9:35, 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:

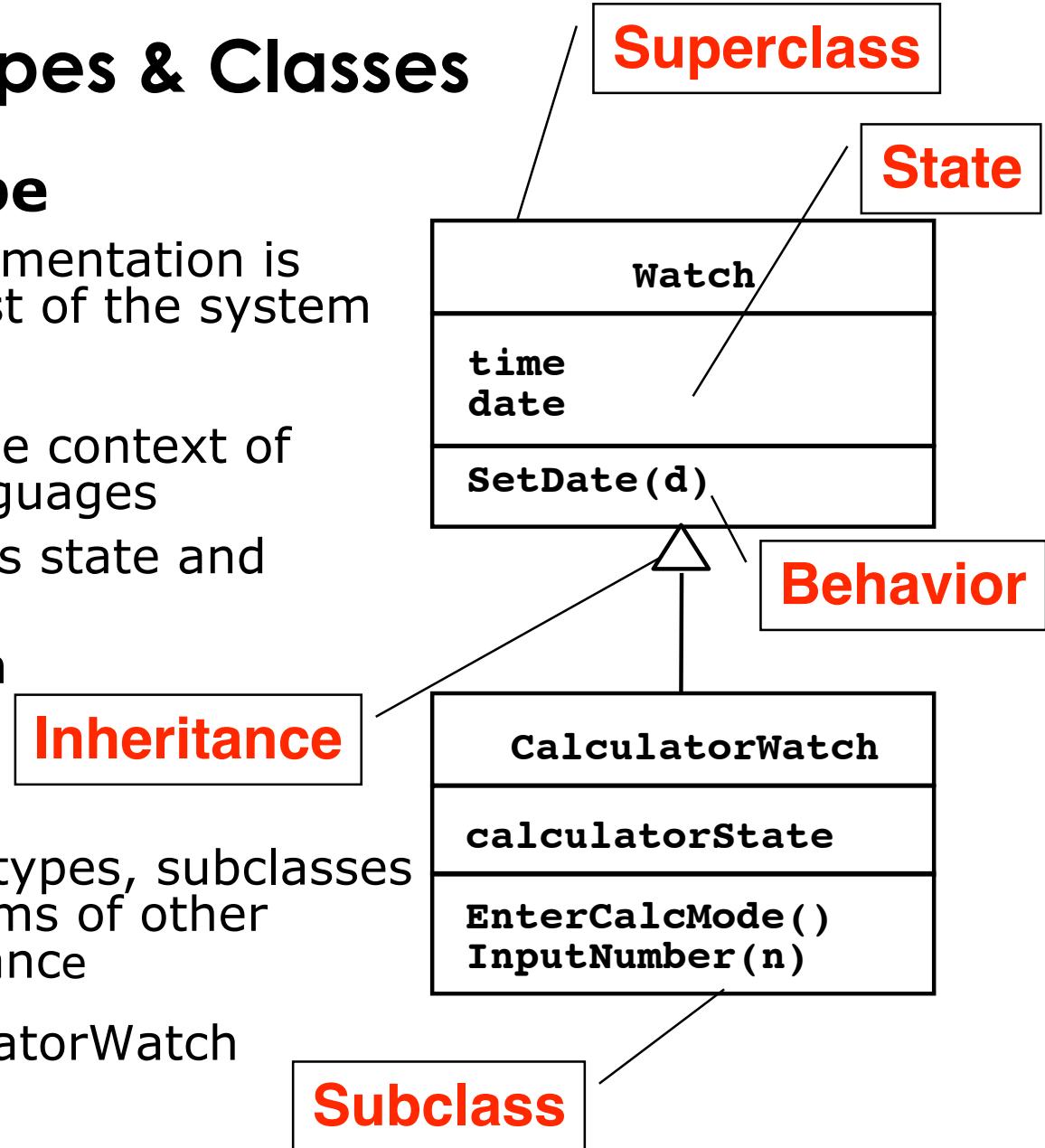
- Development of abstractions to answer specific questions about a set of phenomena while ignoring irrelevant details.

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:**
 - An concept in the context of programming languages
 - Name: int
 - Purpose: integral number
 - Members: 0, -1, 1, 2, -2, ...
- **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 or a subsystem
- A **view** depicts selected aspects of a model
- A **notation** is a set of graphical or textual rules for depicting models and views: formal notations, "real world" representations

System: Airplane

Models:

Flight simulator

Scale model

Views:

Blueprint of

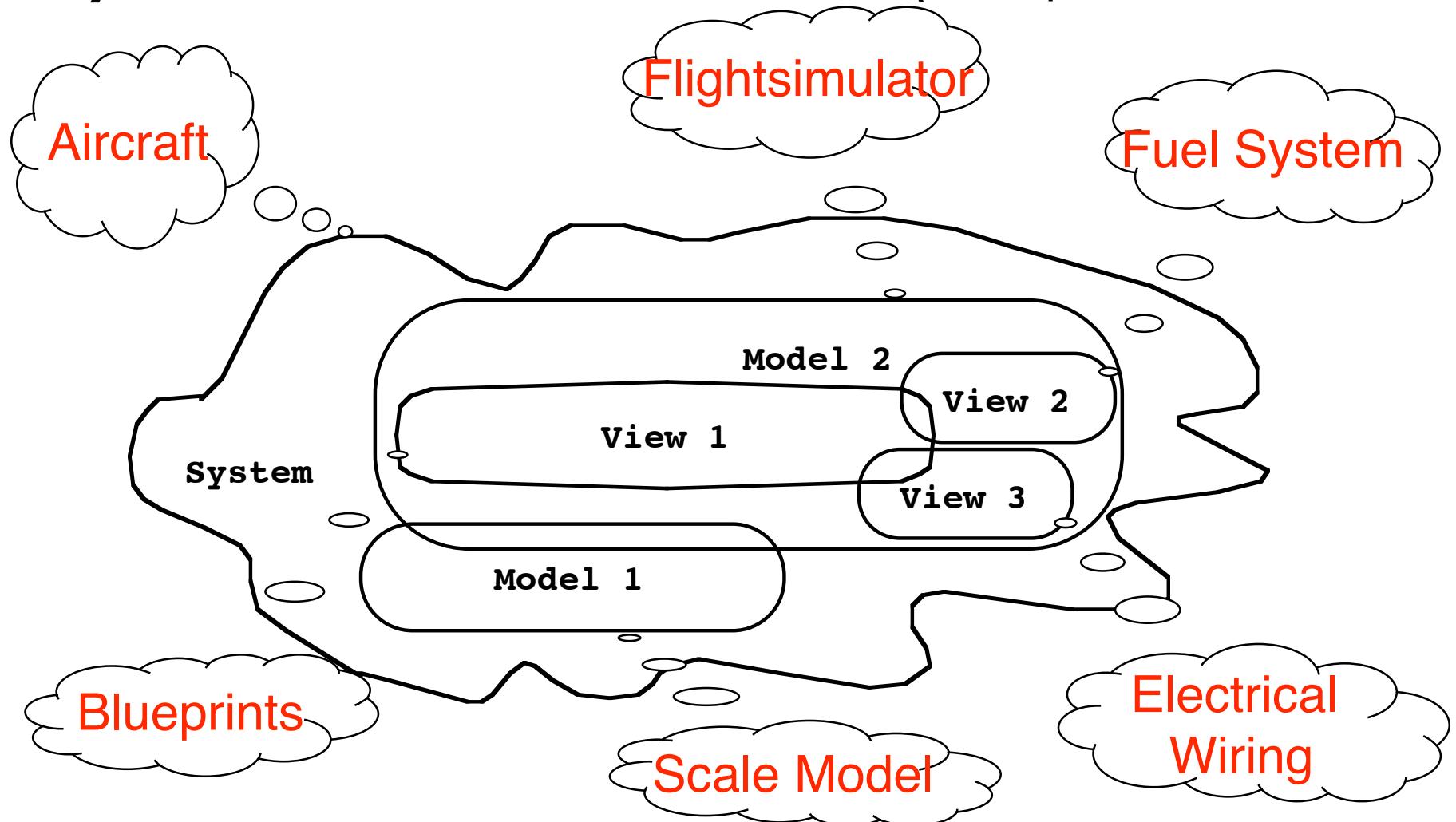
Electrical wir

Fuel system

Sound wave created by airplane

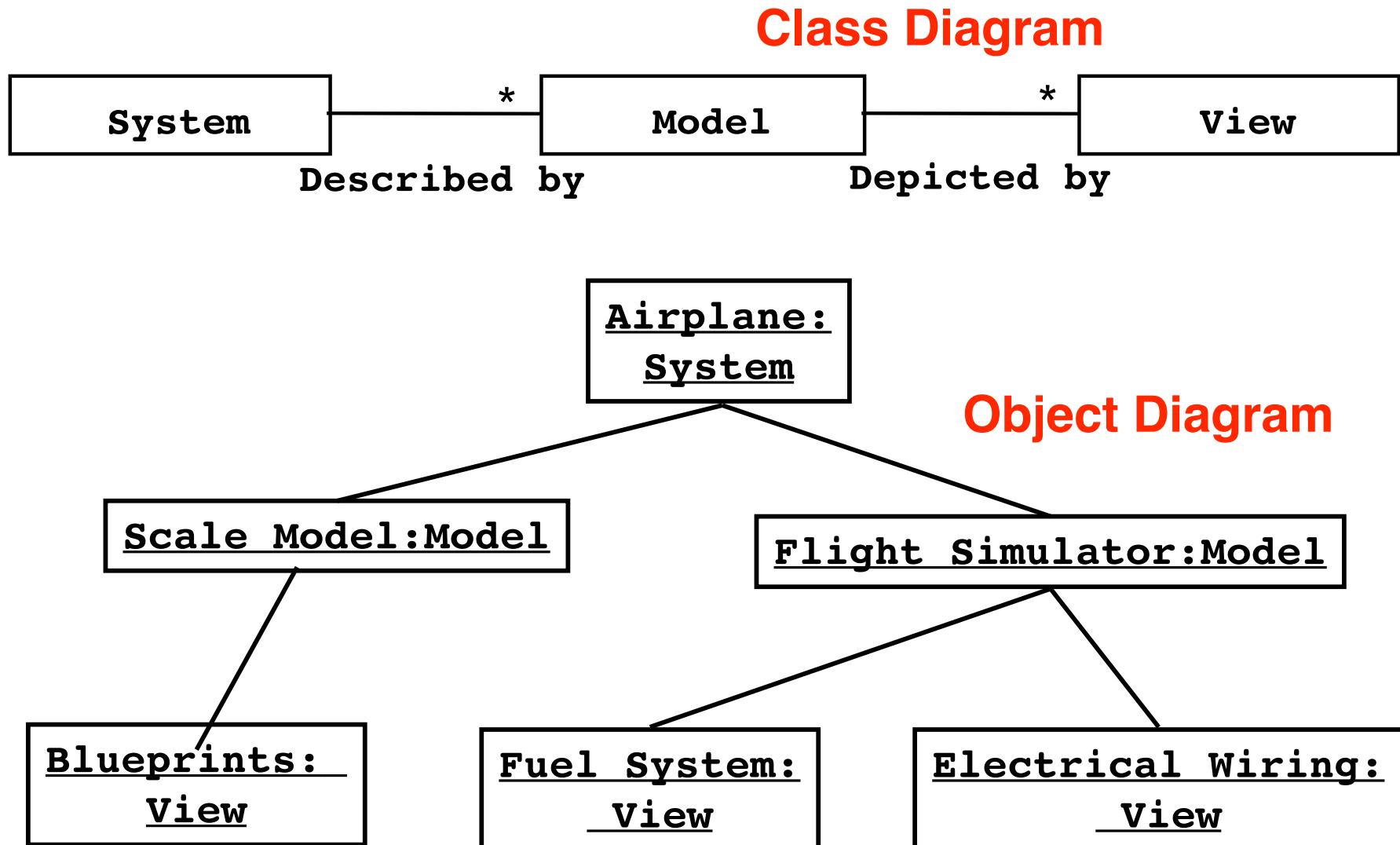


Systems, Models and Views ("Napkin" Notation)



Views and models of a complex system usually overlap

Systems, Models and Views (UML Notation)



Model-Driven Development

1. Build a platform-independent model of an applications functionality and behavior
 - a) Describe model in modeling notation (UML)
 - b) Convert model into platform-specific model
2. Generate executable from platform-specific model

Advantages:

- Code is generated from model ("mostly")
- Portability and interoperability
- Model Driven Architecture effort:
 - <http://www.omg.org/mda/>
- OMG: Object Management Group

Model-driven Software Development

Reality: A stock exchange lists many companies. Each company is identified by a ticker symbol

Analysis results in analysis object model (UML Class Diagram):



Implementation results in source code (Java):

```
public class StockExchange {  
    public m_Company = new Vector();  
};  
public class Company {  
    public int m_tickerSymbol;  
    public Vector m_StockExchange = new Vector();  
};
```

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.

Object-oriented Modeling



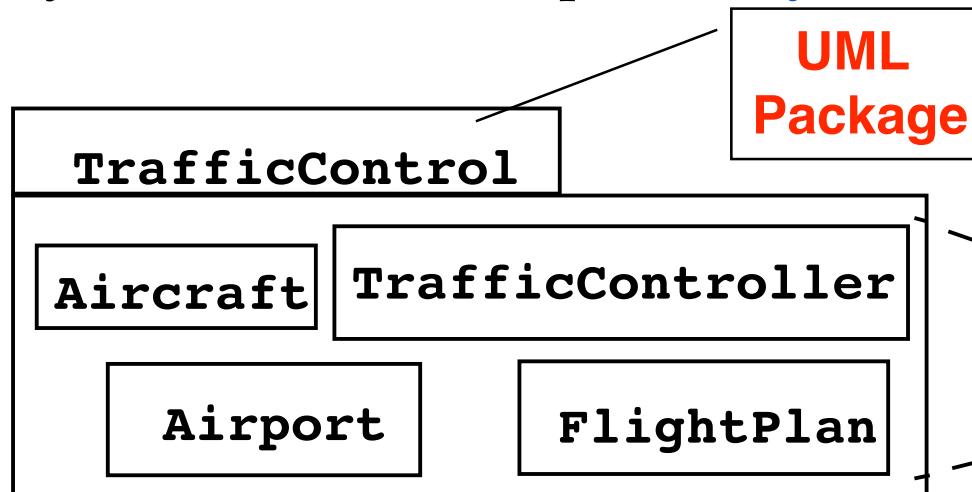
Application Domain
(Phenomena)



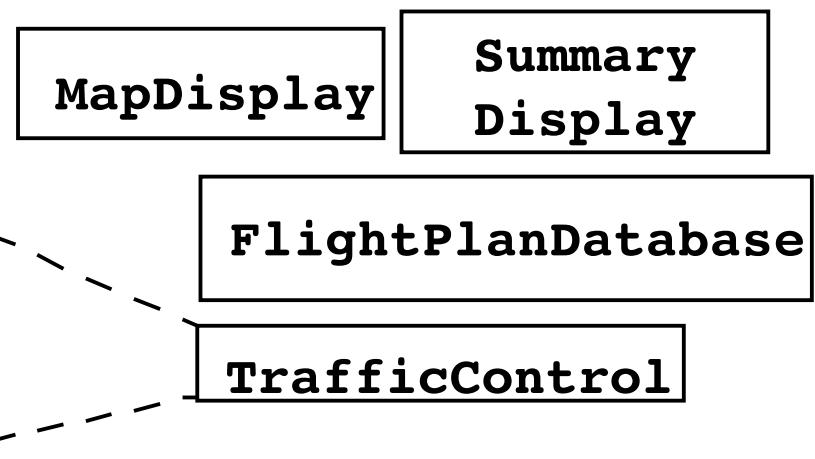
NAME	TIME	PART#	INC #	SIZE(M)	U/M	TIME	NAME	SIZE(M)
INCA	14.24	15.8500	10	INCA	14.24	15.8700		1
INCA	14.24	15.8500	10	INCA	14.24	15.8700		2
BHUT	14.30	15.8500	7	JRCO	14.28	15.9000		3
ISLD	14.30	15.8500	2	ARCA	14.24	15.9000		5
ISLD	14.30	15.8500	2	SHRE	14.22	15.9100		6
LADM	14.18	15.8200	10	REDI	14.24	15.9100		7
NTE	14.22	15.8200	1	SLKC	14.18	15.9400		8
MECH	14.24	15.8200	3	HELT	14.24	15.9500		10
MECH	14.24	15.8200	3	ILTY	14.22	15.9500		1
MECH	14.24	15.8200	3	ISLD	14.20	15.9600		1
LTCO	14.22	15.7500	8	ROFT	14.04	15.9600		1
SCHD	14.19	15.7400	10	SPRT	14.19	15.9800		4
CUST	14.18	15.6800	1	VSBU	13.57	15.9800		5
FEDI	14.24	15.6800	8	VSBU	13.57	16		20
CART	9.00	15.6600	1	OSCO	12.07	16		10
MACH	10.10	15.6600	1	STRD	14.20	16		6

Solution Domain
(Phenomena)

System Model (Concepts) (*Analysis*)



System Model (Concepts) (*Design*)



What is UML?

- UML (Unified Modeling Language)
 - Nonproprietary standard for modeling software systems, OMG
 - Convergence of notations used in object-oriented methods
 - OMT (James Rumbaugh and colleagues)
 - Booch (Grady Booch)
 - OOSE (Ivar Jacobson)
- Current Version 2.0
 - Information at the OMG portal <http://www.uml.org/>
- Commercial tools: Rational (IBM), Together (Borland), Visual Architect (business processes, BCD)
- Open Source tools: ArgoUML, StarUML, Umbrello
- Commercial and Opensource: PoseidonUML (Gentleware)

UML: First Pass

- You can model 80% of most problems by using about 20 % UML
- We teach you those 20%
- 80-20 rule: Pareto principle
 - http://www.ephorie.de/hindle_pareto-prinzip.htm

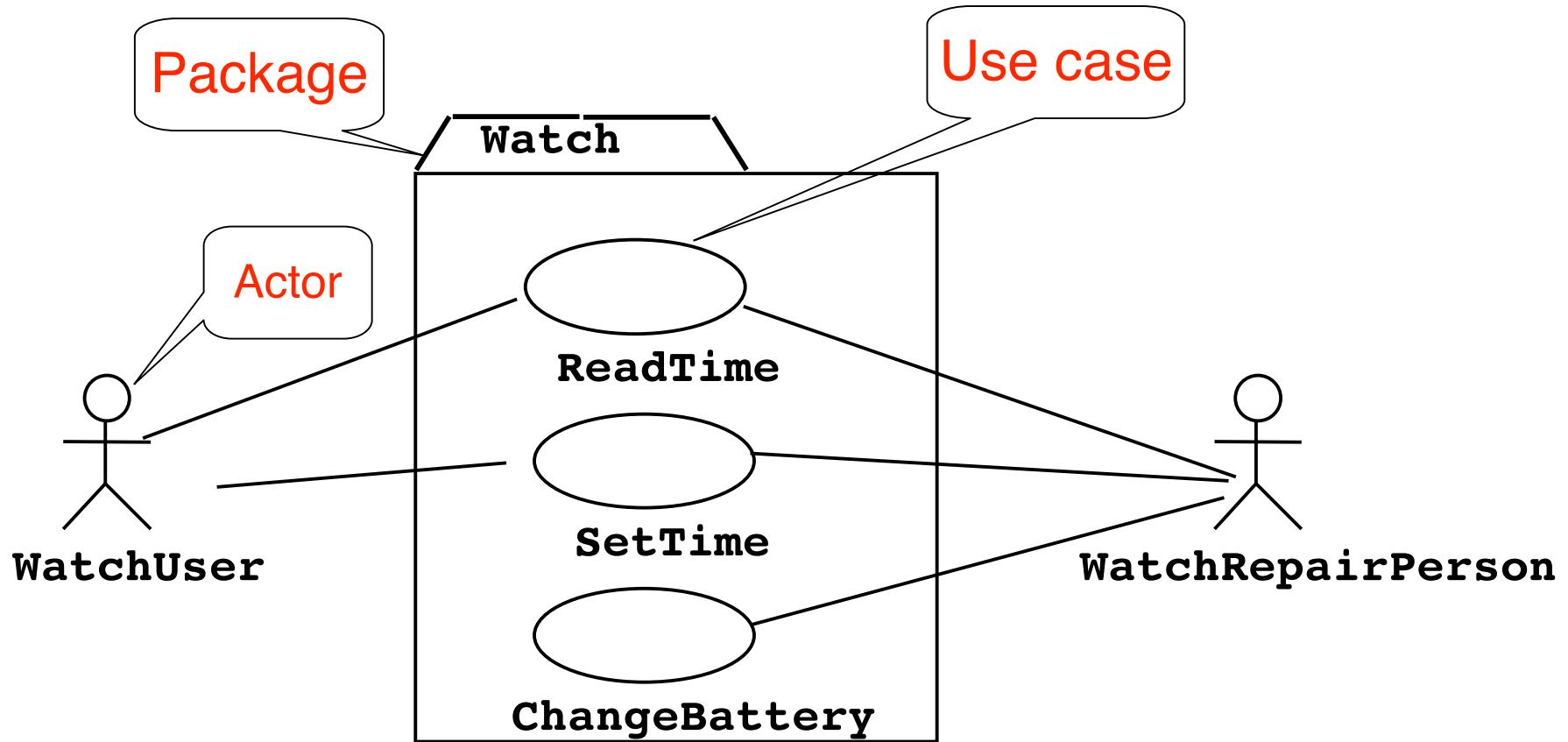
UML First Pass

- **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**
 - Describe the dynamic behavior of an individual object
- **Activity diagrams**
 - Describe the dynamic behavior of a system, in particular the workflow.

UML Core Conventions

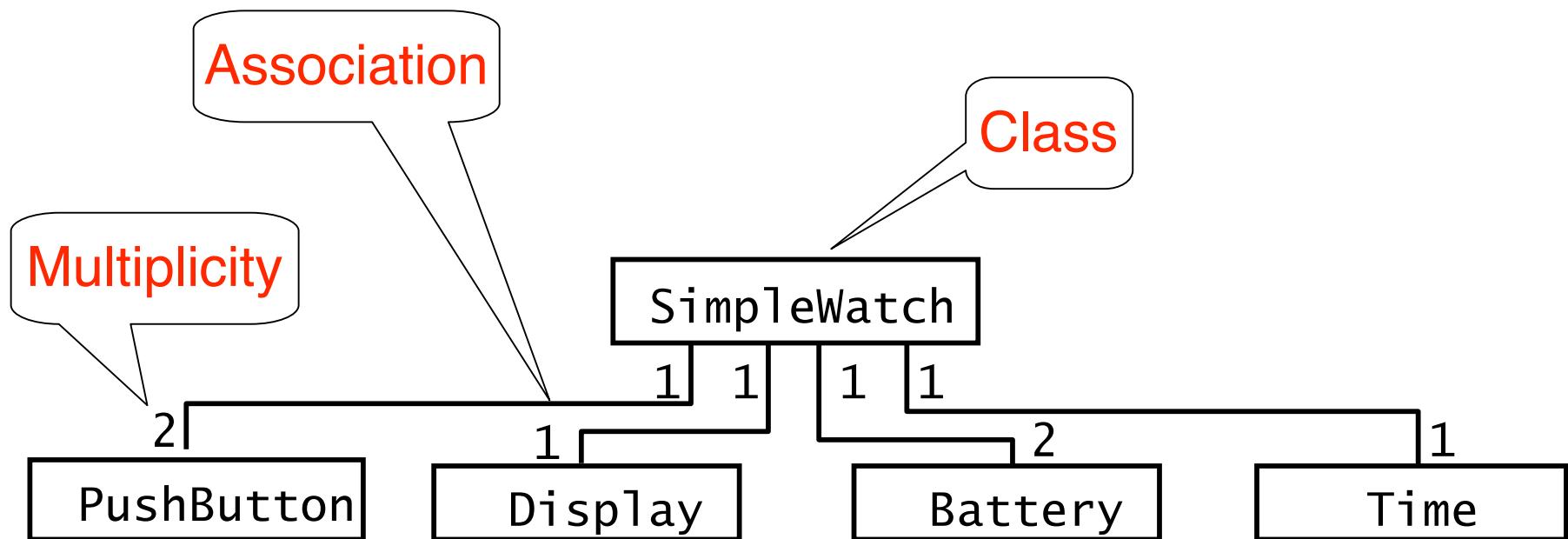
- All UML Diagrams denote graphs of nodes and edges
 - Nodes are entities and drawn as rectangles or ovals
 - Rectangles denote classes or instances
 - Ovals denote functions
- Names of Classes are not underlined
 - SimpleWatch
 - Firefighter
- Names of Instances are underlined
 - myWatch:SimpleWatch
 - Joe:Firefighter
- An edge between two nodes denotes a relationship between the corresponding entities

UML first pass: Use case diagrams



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

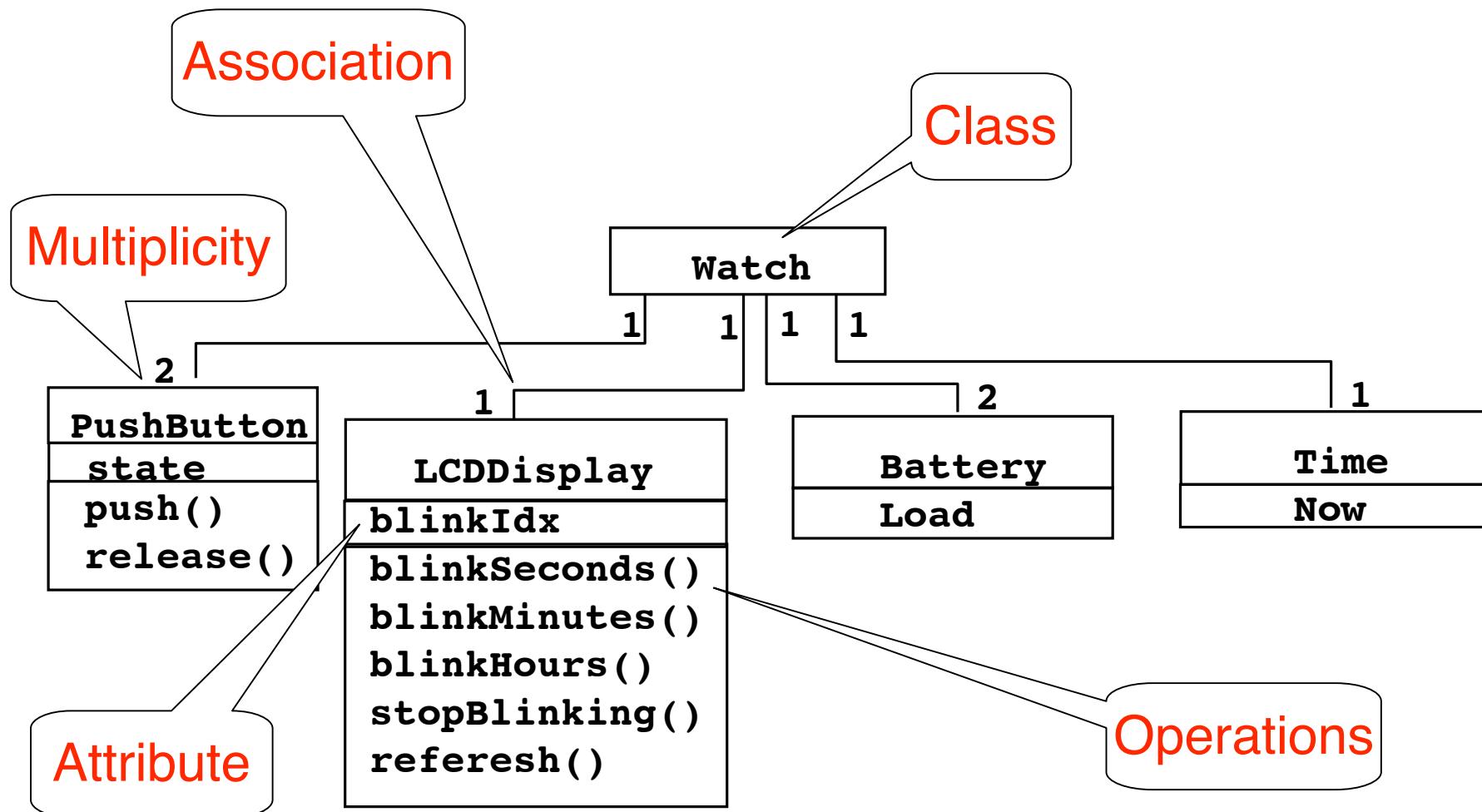
UML first pass: Class diagrams



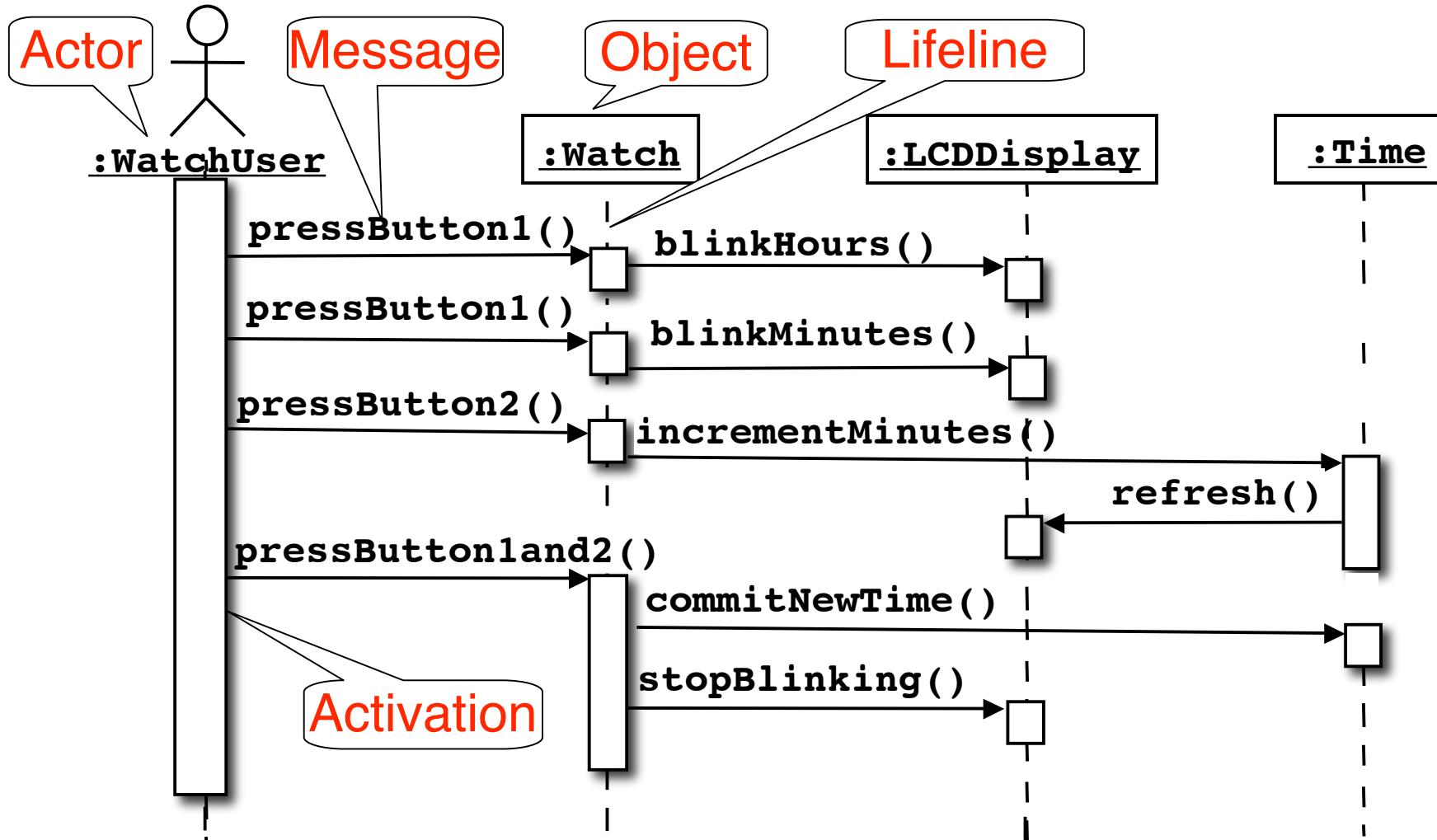
Class diagrams represent the structure of the system

UML first pass: Class diagrams

Class diagrams represent the structure of the system

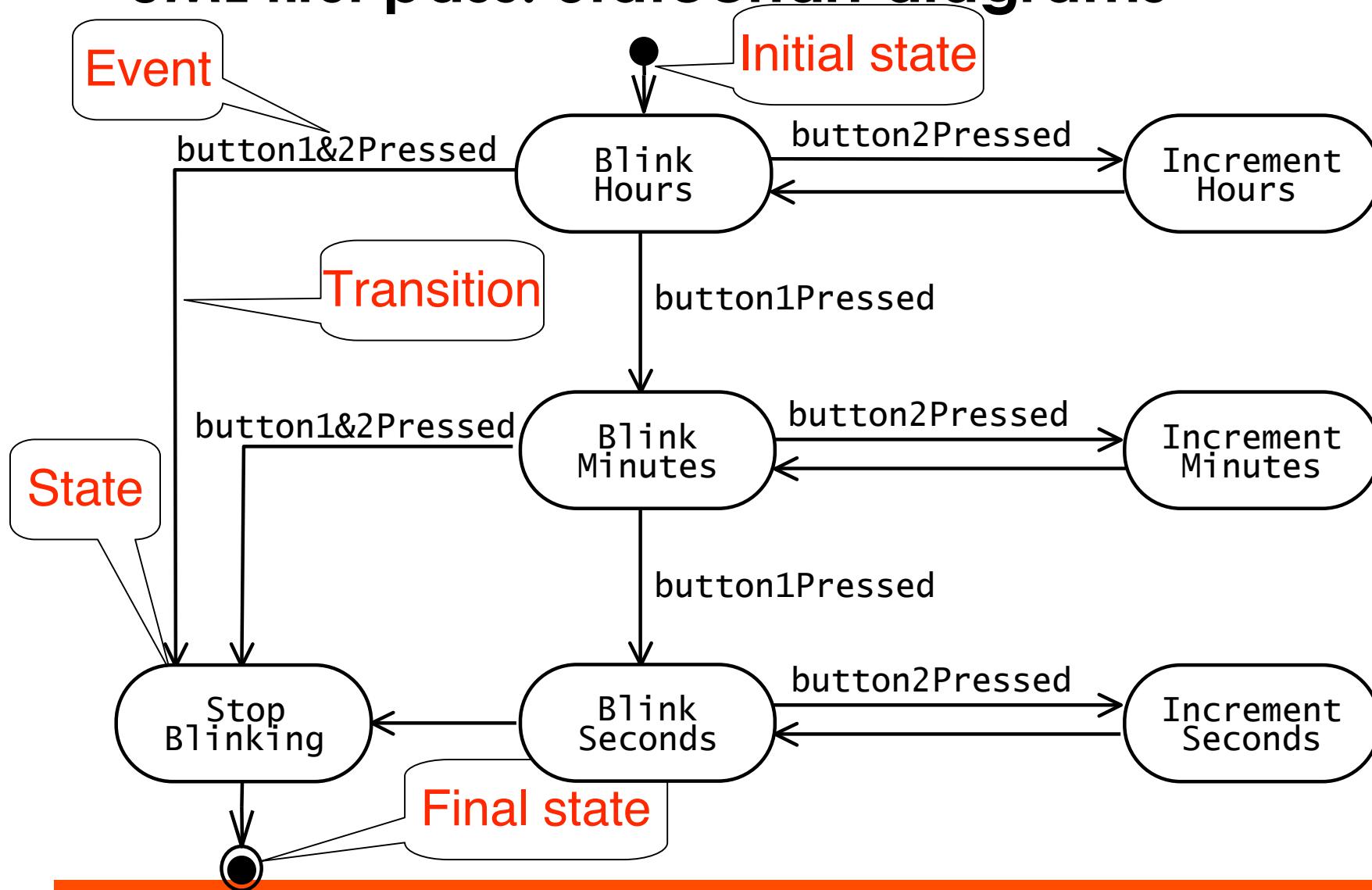


UML first pass: Sequence diagram



Sequence diagrams represent the behavior of a system as messages (“interactions”) between *different objects*

UML first pass: Statechart diagrams



Represent behavior of a *single object* with interesting dynamic behavior.

Other UML Notations

UML provides many other notations

- Activity diagrams for modeling work flows
- Deployment diagrams for modeling configurations (for testing and release management)

What should be done first? Coding or Modeling?

- It all depends....
- Forward Engineering
 - Creation of code from a model
 - Start with modeling
 - Greenfield projects
- Reverse Engineering
 - Creation of a model from existing code
 - Interface or reengineering projects
- Roundtrip Engineering
 - Move constantly between forward and reverse engineering
 - Useful when requirements, technology and schedule are changing frequently.

UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- For now we have concentrated on a few notations:
 - Functional model: Use case diagram
 - Object model: Class diagram
 - Dynamic model: Sequence diagrams, statechart

Additional References

- 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, 1999
- Commercial UML tools
 - Rational Rose XDE for Java
 - <http://www-306.ibm.com/software/awdtools/developer/java/>
 - Together (Eclipse, MS Visual Studio, JBuilder)
 - <http://www.borland.com/us/products/together/index.html>
- Open Source UML tools
 - <http://java-source.net/open-source/uml-modeling>
 - ArgoUML, UMLet, Violet, ...