Introduction

Software Engineering I
WS 2006/2007

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

Intended audience

- Informatik Diplom students (“alte Prüfungsordnung”)
- Informatik Bachelor students (“alte Prüfungsordnung”)
- Computational science and engineering (CSE) students
- Students taking Informatik as a minor (“Nebenfach”)

This lecture is not intended for

- Bachelor students (“neue Prüfungsordnung”)
- Master student (“neue Prüfungsordnung”)
- Students with “Wahlfach Software Engineering”

- If you belong to any of these groups, you must take
  - Software Engineering I: Softwaretechnik by Prof. Broy
  - wwwbroy.in.tum.de/lehre/vorlesungen/sw_technik/WS0607

Objectives of the Class

- Appreciate Software Engineering:
  - Build complex software systems in the context of frequent change
- Understand how to
  - produce a high quality software system within time
  - while dealing with complexity and change
- Acquire technical knowledge
- Acquire basic managerial knowledge

Assumptions for this Class

- Assumption:
  - You are proficient in a programming language,
  - You have no experience in the analysis or design of a system
  - You want to learn more about the technical and managerial aspects of the development of complex software systems
- Beneficial:
  - You have had practical experience with a large software system
  - You have already participated in a large software project
  - You have experienced major problems

Times, Locations and Credits

- Main lecture: MI HS 1, 00.02.001
  - Tuesdays 12:15-13:45
  - Wednesdays 9:15-10:00
- Exercises: Miniproject
  - Scheduled for January
- Written Exams:
  - Mid-term: Dec 20, 2006, Wednesday 9:00-10:30
    - Location to be announced
  - Final: Time and Location to be announced

Grading and Credits: Bachelor Students

- Exercises: 20 %
  - Mini-Project
- Mid-term: 30 %
- Final: 50 %

- Area: Informatics
- Hours per week: 3 lectures + 2 exercises
- ECTS Credits: 6

Acquire Technical Knowledge

- Understand system modeling
- Learn a modeling notation (Unified Modeling Language UML)
- Learn different modeling methods
- Learn how to use Tools
- Testing
- Model-based software development

Acquire Basic Managerial Knowledge

- Software Project Management
- Software Lifecycle
- Rationale Management
- Configuration Management
- Methodologies

- Expansion on these topics:
  - Course Software Engineering II in the Summer

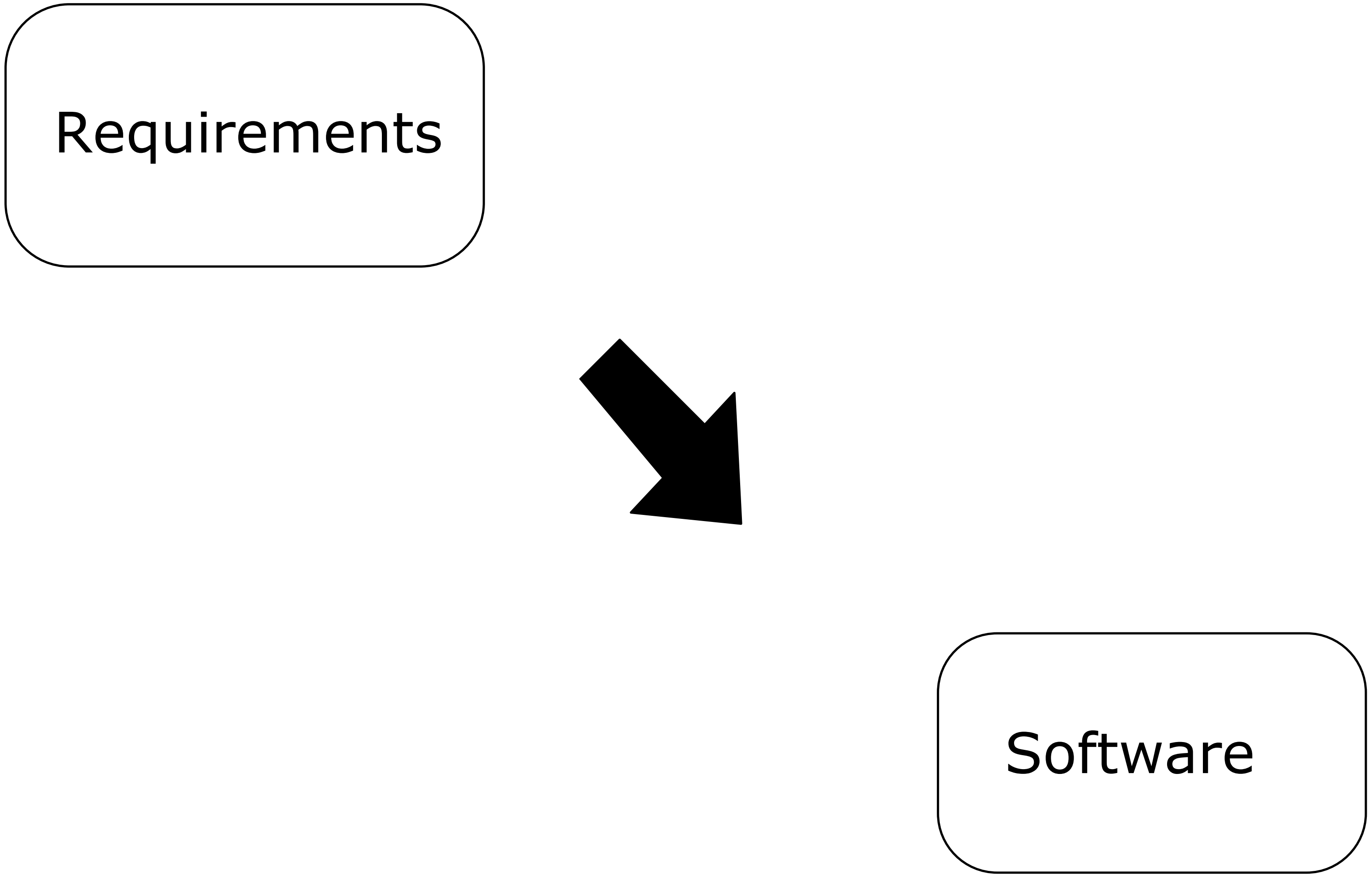
Outline of Today's Lecture

- High quality software: State of the art
- Modeling complex systems
- Dealing with change
- Concepts
  - Abstraction
  - Modeling
  - Hierarchy
- Organizational issues
  - Lecture schedule
  - Exercise schedule
  - Associated Project

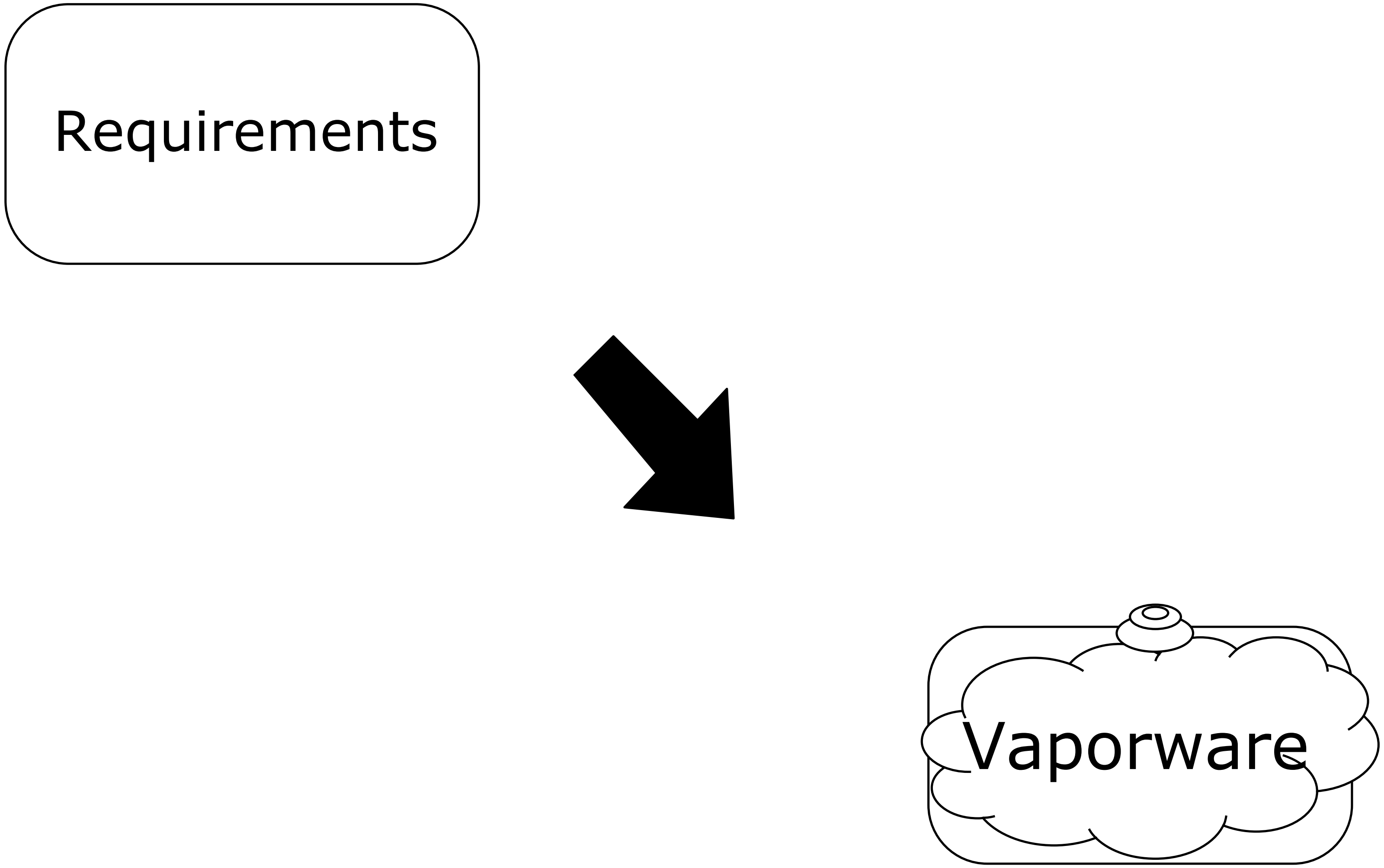
Software Production has Poor Track Record

- **Example:** Space Shuttle Software
- **Cost:** $10 Billion, millions of dollars more than planned
- **Time:** 3 years late
- **Quality:** First launch of Columbia was cancelled
  - Synchronization problem with the Shuttle's 5 onboard computers.
- **Substantial errors still exist**
  - Astronauts are supplied with a book of known software problems "Program Notes and Waivers".

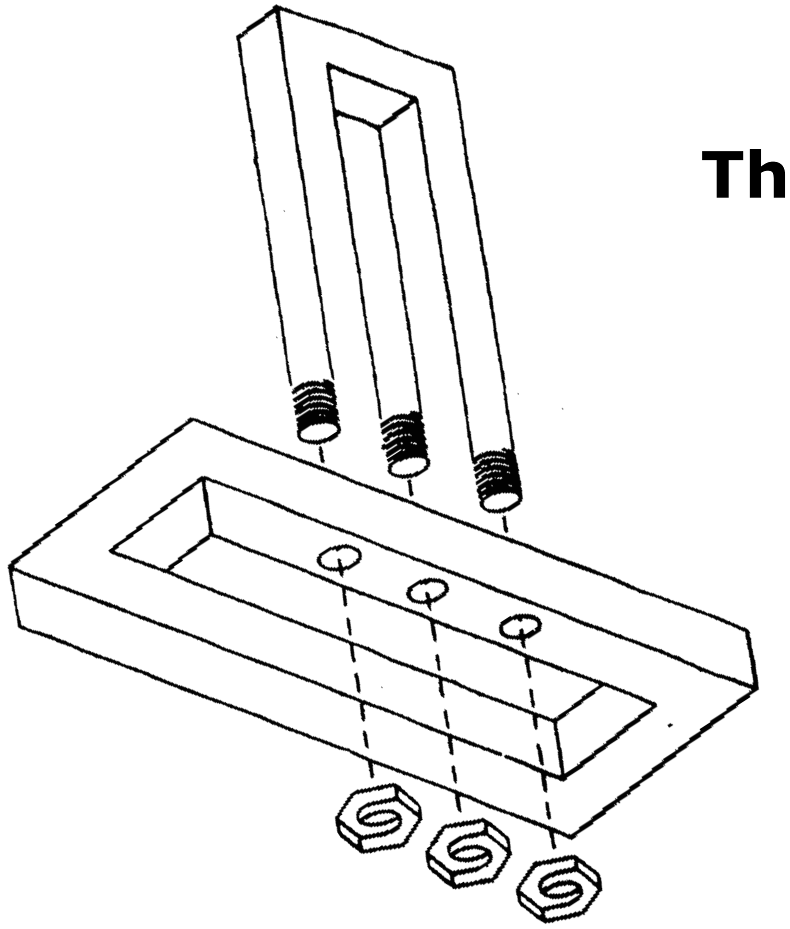
Limitations of Non-engineered Software



Limitations of Non-engineered Software



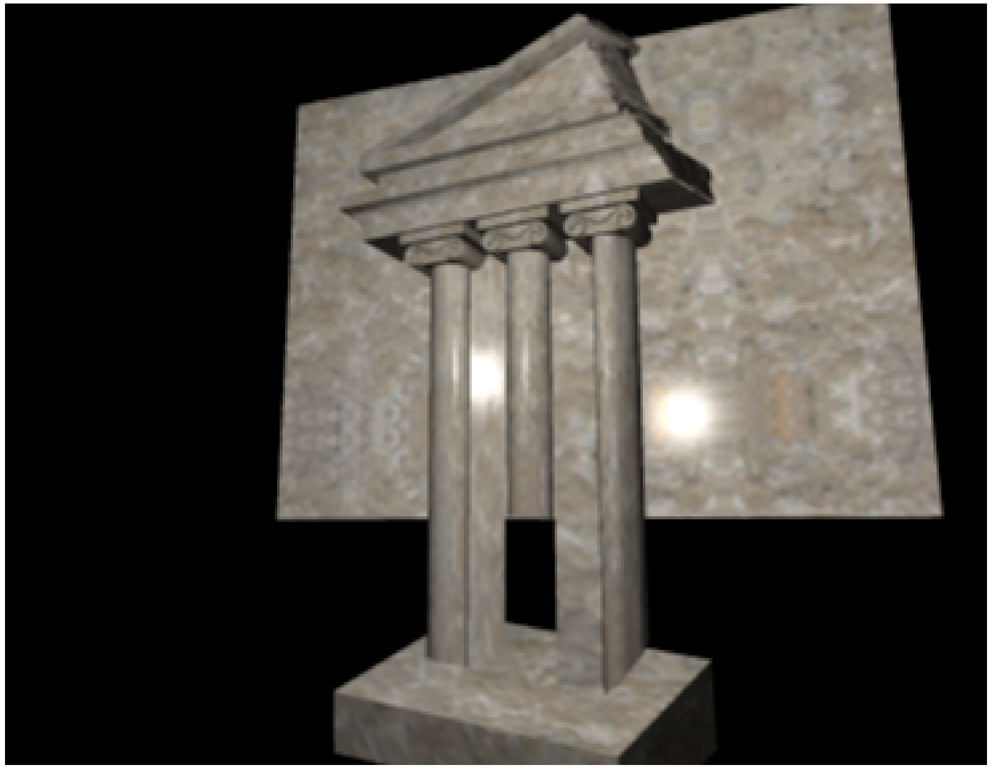
Can you develop this system?



The impossible Fork

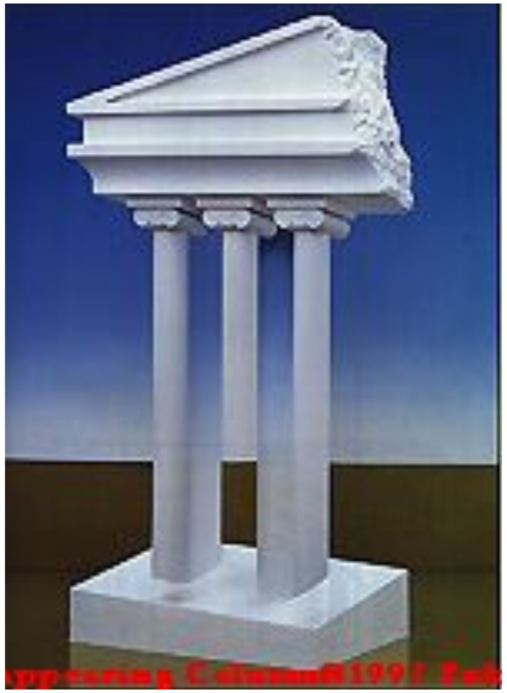
Physical Model of the impossible Fork (Shigeo Fukuda)

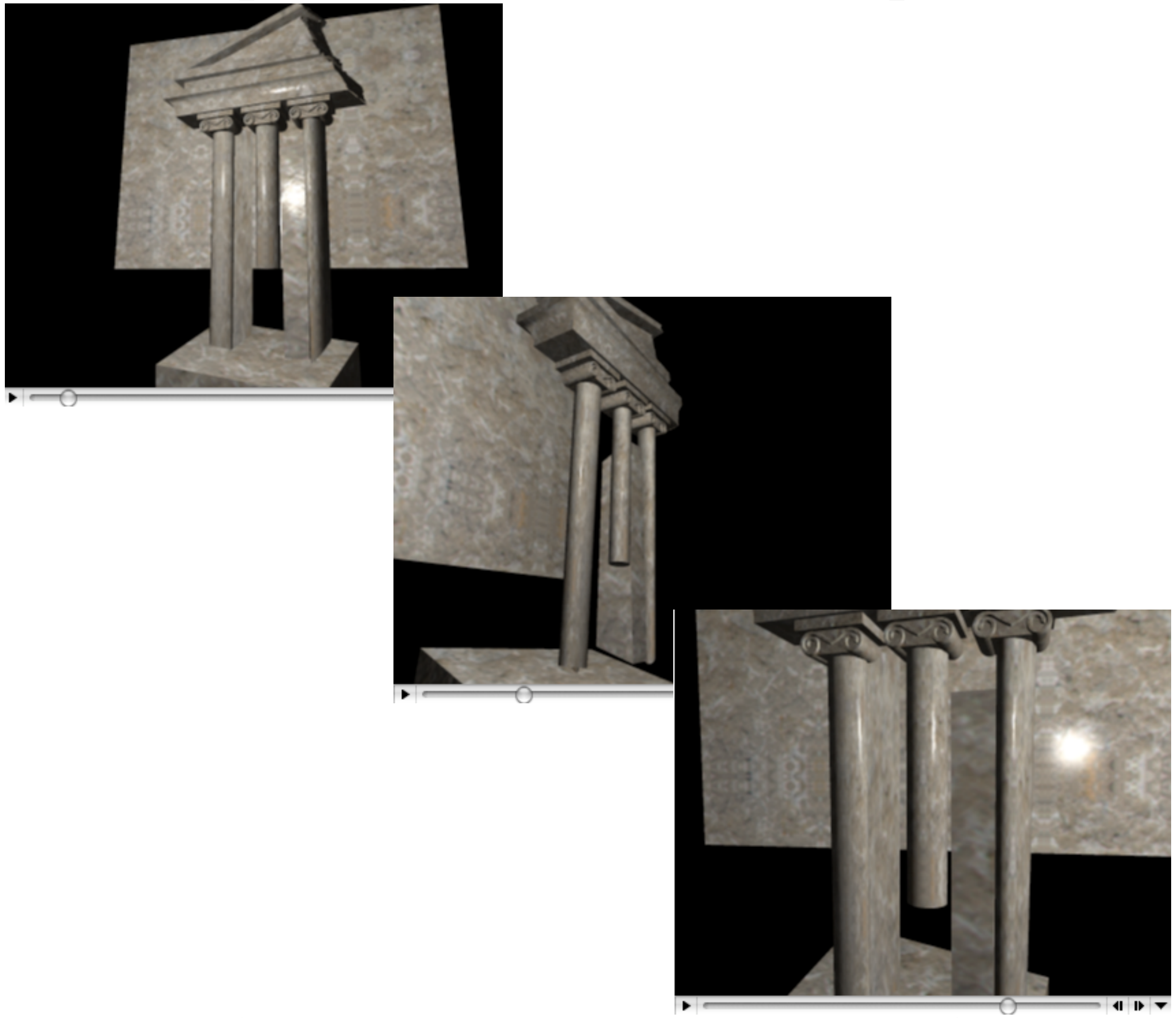
ppearing Column©1997 Fuk



Source http://neuro.caltech.edu/~seckel/mod/movies/fukuda/DisappearingColumn.mov

Physical Model of the impossible Fork (Shigeo Fukuda)





Source http://neuro.caltech.edu/~seckel/mod/movies/fukuda/DisappearingColumn.mov

Why are software systems complex?

- The problem domain is difficult
- The development process is very difficult to manage
- Software offers extreme flexibility
- Software is a discrete system
    - Continuous systems have no hidden surprises (Parnas)
    - Discrete systems can have hidden surprises!

Software Engineering is more than writing Code

- Problem solving
  - Creating a solution
  - Engineering a system based on the solution
- Modeling
- Knowledge acquisition
- Rationale management

Techniques, Methodologies and Tools

- **Techniques:**
  - Formal procedures for producing results using some well-defined notation
- **Methodologies:**
  - Collection of techniques applied across software development and unified by a philosophical approach
- **Tools:**
  - Instruments or automated systems to accomplish a technique
  - CASE = Computer Aided Software Engineering

Computer Science vs SoftwareEngineering

- Computer Scientist
    - Assumes techniques and tools have to be developed.
    - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
    - Has infinite time…

Computer Science vs Software Engineering (cont'd)

- Engineer
  - Develops a solution for a problem in an application domain for a client
  - Uses computers & languages, techniques and tools
- Software Engineer
  - Works in multiple application domains
  - Has only 3 months...
  - …while changes occurs in requirements and available technology

Software Engineering: A Working Definition

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

a *high quality software* system
with a given *budget*
before a given *deadline*
while *change* occurs

Challenge: Dealing with complexity and change

Software Engineering: A Problem Solving Activity

- **Analysis:**
  - Understand the nature of the problem and break the problem into pieces
- **Synthesis:**
  - Put the pieces together into a large structure

For problem solving we use techniques, methodologies and tools

Course Outline

Dealing with Complexity

- Modeling
- UML Notation
- Requirements Elicitation
- Requirements Analysis
- System Design
- Detailed Design
- Implementation & Testing

Dealing with Change

- Rationale Management
- Configuration Management
- Software Project Management
- Software Life Cycle
- Methodologies

Application of these Concepts: Project

Lecture Outline

Lecture Outline (cont'd)

System Design:

10. Design Goals & System Decomposition
11. Architectural Styles
12. Addressing Design Goals

Object Design:

13. Reuse
14. Basic Design Patterns
15. Advanced Design Patterns
16. Object Constraint Language OCL
17. Interface Specification

Implementation:

18. Mapping Object Models to Java Code
19. Mapping Object Models to Relational Schema

Lecture Outline (cont'd)

Testing:

- 20. Unit Testing
- 21. System and Usability Testing

Configuration Management:

- 22. Basic Concepts
- 23. Configuration Management Tools
- 24. Build Management

Software Lifecycle

- 25. Software Lifecycle Modeling

Tentative Lecture Schedule

Subject to Change!

Tuesdays 12:15-13:45

✓ Oct 24: Introduction

- Oct 31: Modeling with UML
- Nov 7: Project Organization & Communication
- Nov 14: Functional Modeling
- Nov 21: Dynamic Modeling
- Nov 28: Architectural Styles
- Nov 30: Reuse
- Dec 5: No lecture
- Dec 12: Design Patterns
- Dec 19: Object Constraint Language

Wednesday 9:15-10:00

- Oct 25: Introduction ctd

- Nov 1: Holiday (Allerheiligen)
- Nov 8: Requirements Elicitation
- Nov 15: Object Modeling
- Nov 22: Design Goals
- Nov 29: Addressing Design Goals
- Dec 6: No lecture
- Dec 13: Interface Specification
- Dec 20: Mid-term

Lecture Plan for January/February 2007

Tuesdays 13:15-14:15

- Jan 9: Unit Testing
- Jan 16: System Testing
- Jan 23: Configuration Management
- January 30: Software Lifecycle
- Feb 7: Miniproject Presentations

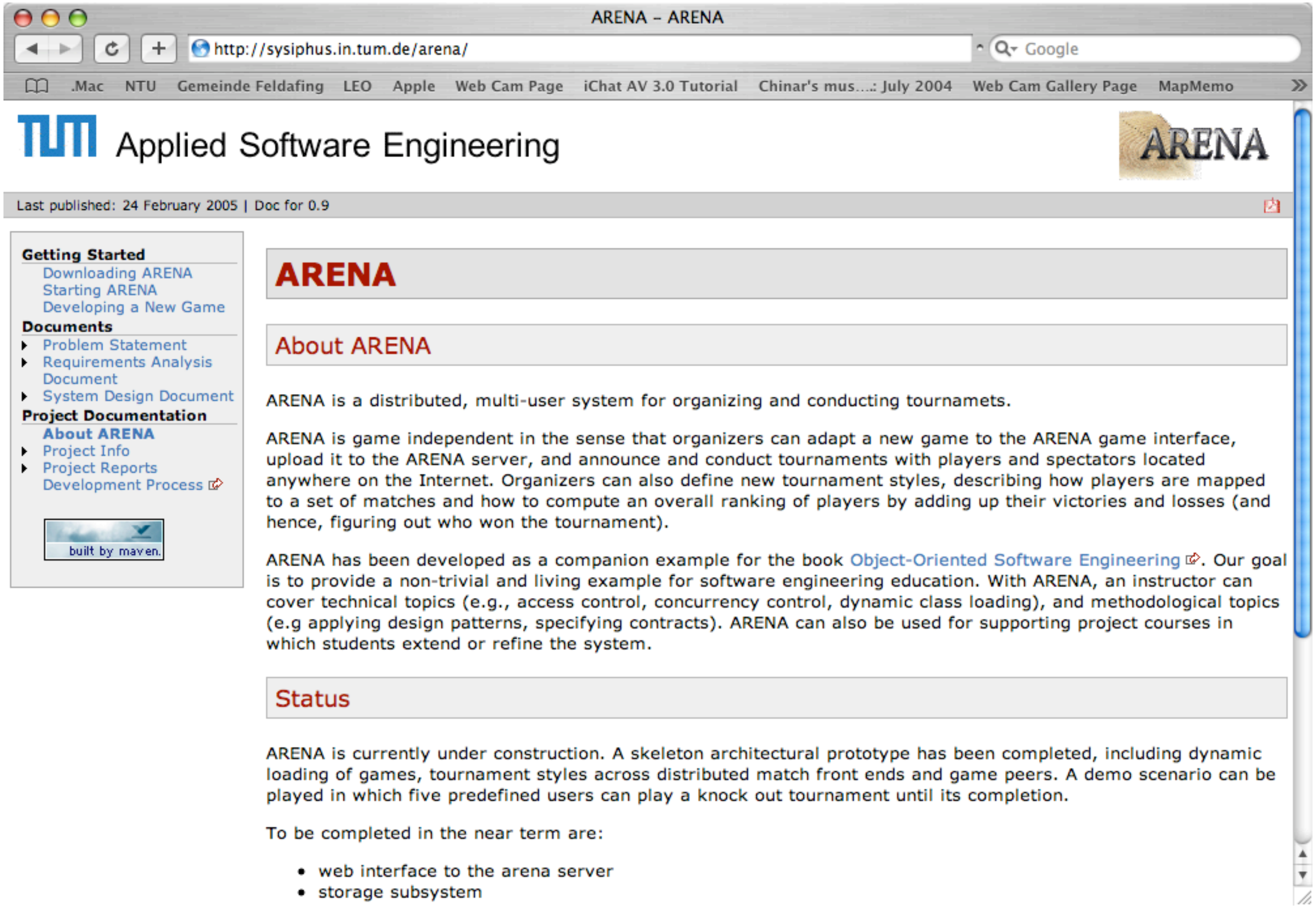
Wednesday 9:15-10:00

- Jan 10: Integration Testing
- Jan 17: Build Management
- Jan 24: Software lifecycle
- Jan 31: Guest lecture
- Feb 8: Miniproject Presentations II

- Final Exam: To be announced
- Tools: Subversion (Configuration Management), Maven (Web-Site Generation), Ant, Cruise-Control (Build Management()

Case Study: ARENA

- This project will be used in the lectures to illustrate software engineering concepts and artifacts
- ARENA specific models and documents will be made available incrementally during the course
- ARENA’s source code is available
  - http://sysiphus.in.tum.de/arena


ARENA – ARENA
http://sysiphus.in.tum.de/arena/
Google
.Mac NTU Gemeinde Feldafing LEO Apple Web Cam Page iChat AV 3.0 Tutorial Chinar's mus....: July 2004 Web Cam Gallery Page MapMemo

Applied Software Engineering

ARENA

Last published: 24 February 2005 | Doc for 0.9

Getting Started
- Downloading ARENA
- Starting ARENA
- Developing a New Game

Documents
- Problem Statement
- Requirements Analysis Document
- System Design Document

Project Documentation
- **About ARENA**
- Project Info
- Project Reports
- Development Process

ARENA

About ARENA

ARENA is a distributed, multi-user system for organizing and conducting tournamets.

ARENA is game independent in the sense that organizers can adapt a new game to the ARENA game interface, upload it to the ARENA server, and announce and conduct tournaments with players and spectators located anywhere on the Internet. Organizers can also define new tournament styles, describing how players are mapped to a set of matches and how to compute an overall ranking of players by adding up their victories and losses (and hence, figuring out who won the tournament).

ARENA has been developed as a companion example for the book Object-Oriented Software Engineering. Our goal is to provide a non-trivial and living example for software engineering education. With ARENA, an instructor can cover technical topics (e.g., access control, concurrency control, dynamic class loading), and methodological topics (e.g applying design patterns, specifying contracts). ARENA can also be used for supporting project courses in which students extend or refine the system.

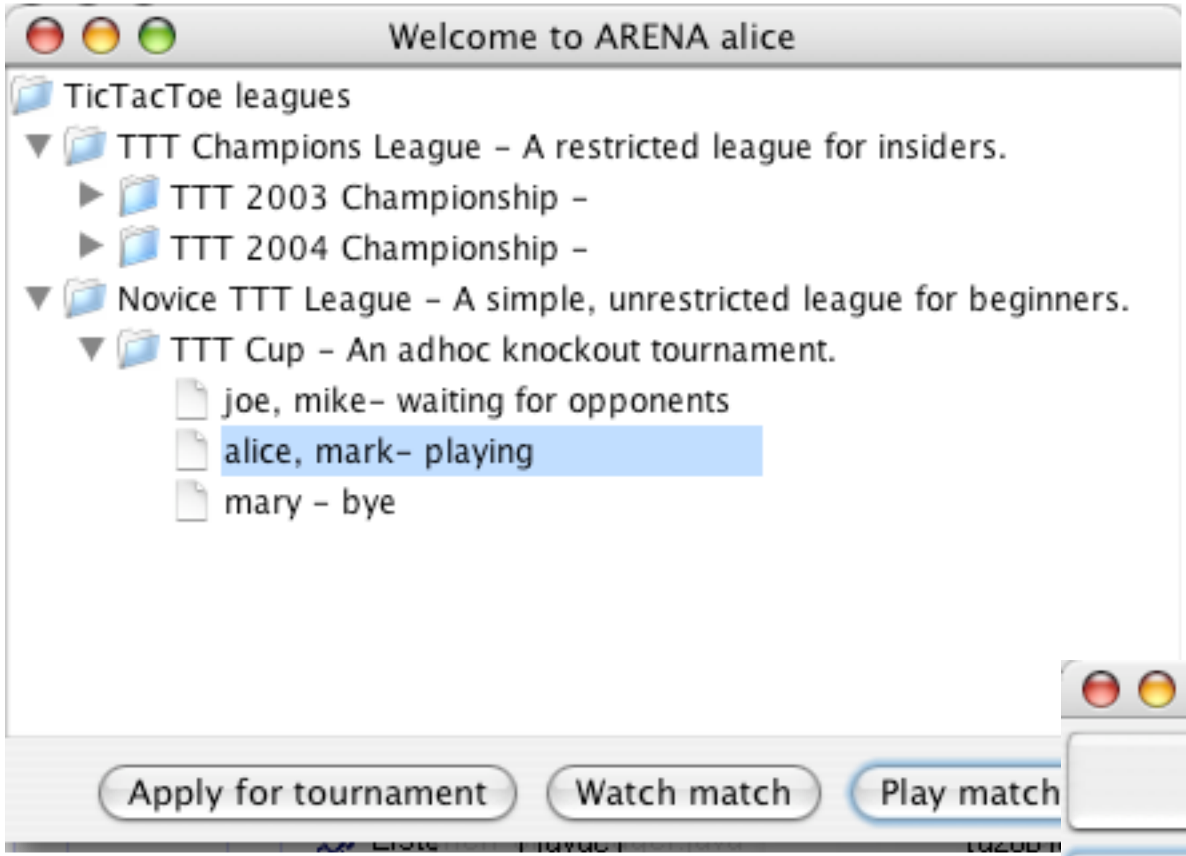
Status

ARENA is currently under construction. A skeleton architectural prototype has been completed, including dynamic loading of games, tournament styles across distributed match front ends and game peers. A demo scenario can be played in which five predefined users can play a knock out tournament until its completion.

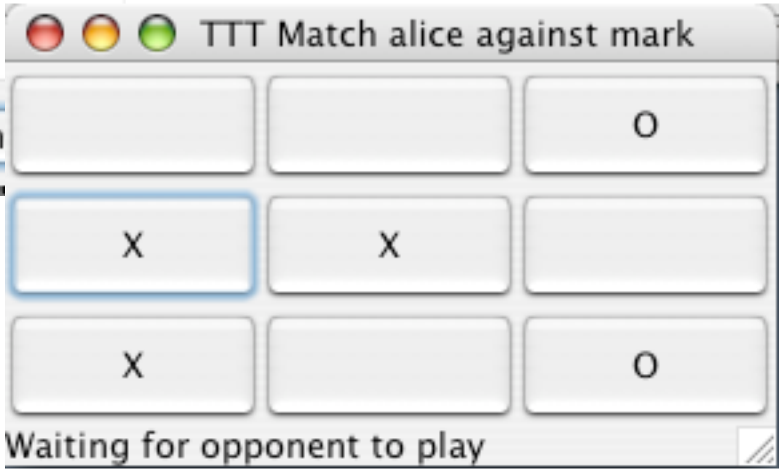
To be completed in the near term are:

- web interface to the arena server
- storage subsystem

Playing TicTacToe within ARENA


Welcome to ARENA alice

TicTacToe leagues
- TTT Champions League – A restricted league for insiders.
  - TTT 2003 Championship –
  - TTT 2004 Championship –
- Novice TTT League – A simple, unrestricted league for beginners.
  - TTT Cup – An adhoc knockout tournament.
    - joe, mike– waiting for opponents
    - alice, mark– playing
    - mary – bye


TTT Match alice against mark

		O
X	X	
X		O

Waiting for opponent to play

Mini Project: Asteroids

- Main Goal:
  - Practice and apply the important concepts of the lectures
  - Become proficient in using and applying these concepts
- Context
  - Asteroids Game
  - Examples of an Asteroids Implementation:
    - http://www.surfnetkids.com/games/asteroids-game.htm
- Project Tasks:
  - Addition of nonfunctional requirements to an existing implementation of Asteroids using model-based development techniques and design patterns
  - Integration of Asteroids with ARENA

Textbook

- Bernd Bruegge, Allen H. Dutoit:
  - **Object-Oriented Software Engineering: Using UML, Design Patterns and Java**, 2nd edition, Prentice Hall, September 2003
- German Version:
  - Bernd Brügge, Allen H. Dutoit: "Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java, Pearson Education, Oktober 2004
- You can get a 10% discount for the english edition, if you order from this URL
  - www.pearson-studium.de/main/main.asp?page=bookdetails&ProductID=111686

Additional Readings

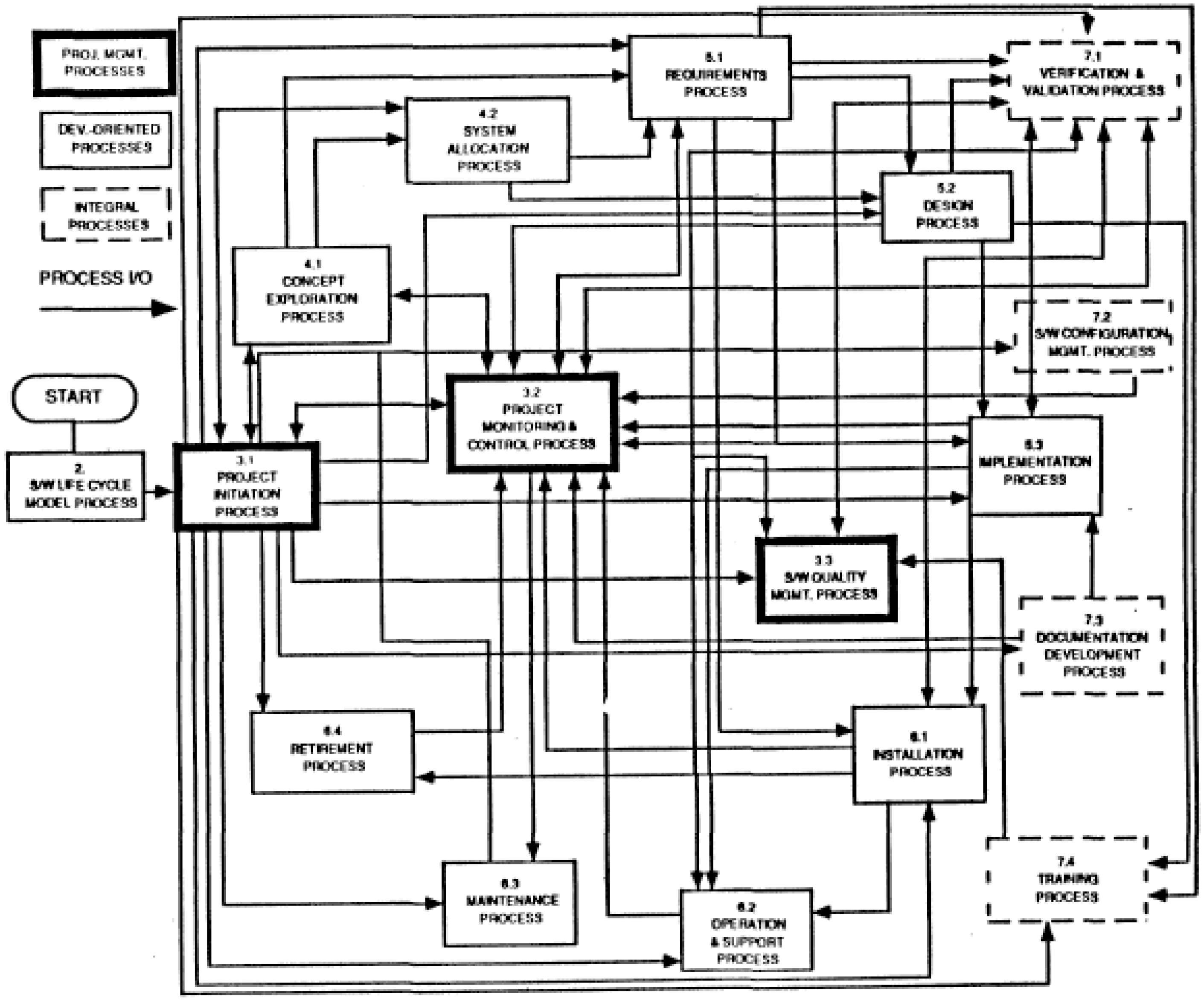
- Additional readings are announced for each lecture

- Additional Readings for this lecture:
  - K. Popper, “Objective Knowledge, an Evolutionary Approach”, Oxford Press, 1979.

  - Falsification

Let's start bruegge@in.tum.de Subject: SE 1 or Software Engineering 1

What is this?


PROJ. MGMT. PROCESSES
DEV.-ORIENTED PROCESSES
INTEGRAL PROCESSES
PROCESS I/O
START
2. S/W LIFE CYCLE MODEL PROCESS
3.1 PROJECT INITIATION PROCESS
3.2 PROJECT MONITORING & CONTROL PROCESS
3.3 S/W QUALITY MGMT. PROCESS
4.1 CONCEPT EXPLORATION PROCESS
4.2 SYSTEM ALLOCATION PROCESS
5.1 REQUIREMENTS PROCESS
5.2 DESIGN PROCESS
5.3 IMPLEMENTATION PROCESS
6.1 INSTALLATION PROCESS
6.2 OPERATION & SUPPORT PROCESS
6.3 MAINTENANCE PROCESS
6.4 RETIREMENT PROCESS
7.1 VERIFICATION & VALIDATION PROCESS
7.2 S/W CONFIGURATION MGMT. PROCESS
7.3 DOCUMENTATION DEVELOPMENT PROCESS
7.4 TRAINING PROCESS

3 Ways to deal with Complexity

1. Abstraction
2. Decomposition
3. Hierarchy

Abstraction 10 24 2006

- 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
  - Chunking: Group collection of objects
- Abstraction allows us to ignore unessential details
- Two definitions for abstraction:
  - **Abstraction as activity:** Abstraction is a thought process where ideas are distanced from objects
  - **Abstraction as entity:** Abstraction is the resulting idea of a thought process where an idea is distanced from an object
- 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 can use models to describe Software Systems

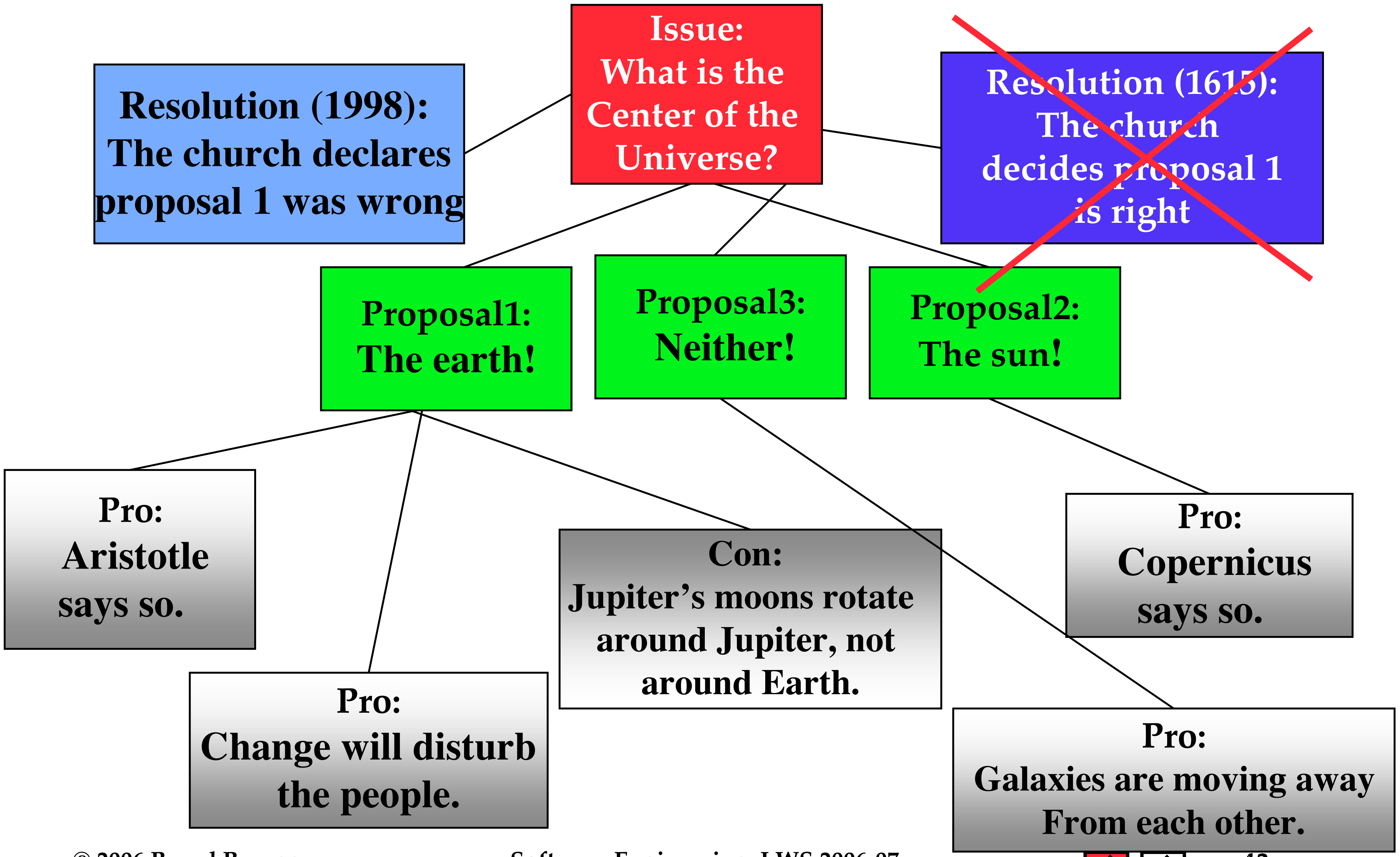
- Object model: What is the structure of the system?
- Functional model: What are the functions of the system?
- Dynamic model: How does the system react to external events?

- 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: What is the Center of the Universe?
Resolution (1998): The church declares proposal 1 was wrong
~~Resolution (1615): The church decides proposal 1 is right~~
Proposal1: The earth!
Proposal3: Neither!
Proposal2: The sun!
Pro: Aristotle says so.
Pro: Change will disturb the people.
Con: Jupiter's moons rotate around Jupiter, not around Earth.
Pro: Copernicus says so.
Pro: Galaxies are moving away From each other.

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.

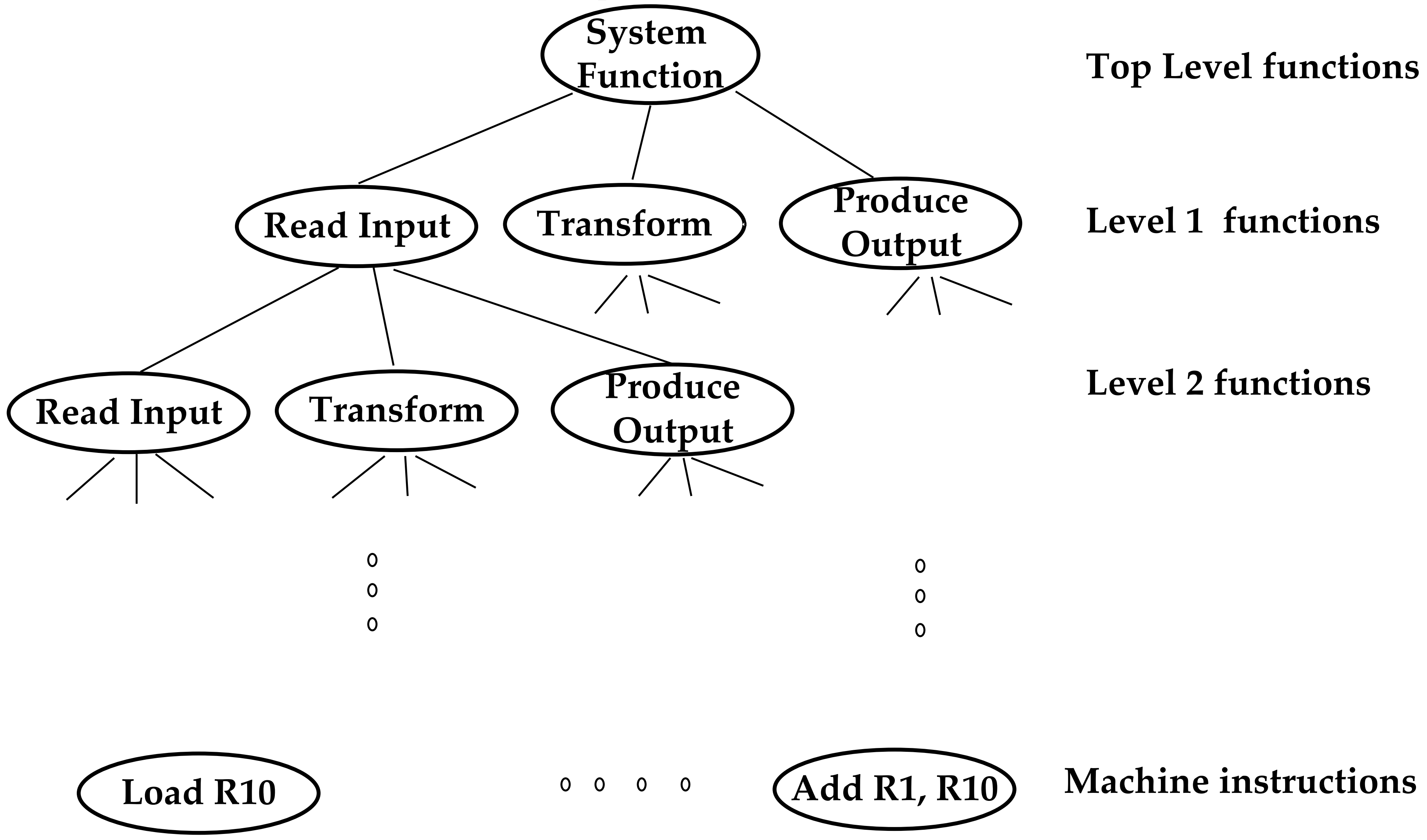
Decomposition (cont'd)

- 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



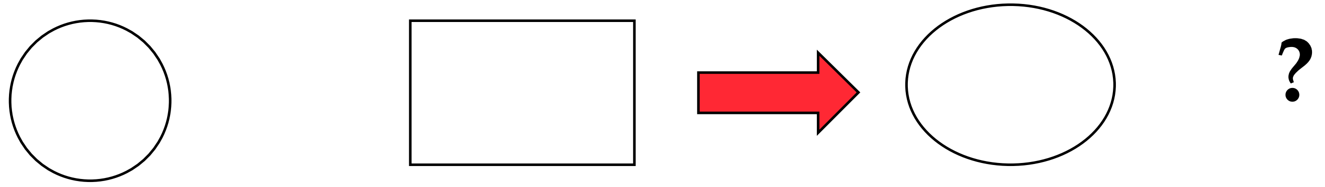
Functional Decomposition

- 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.

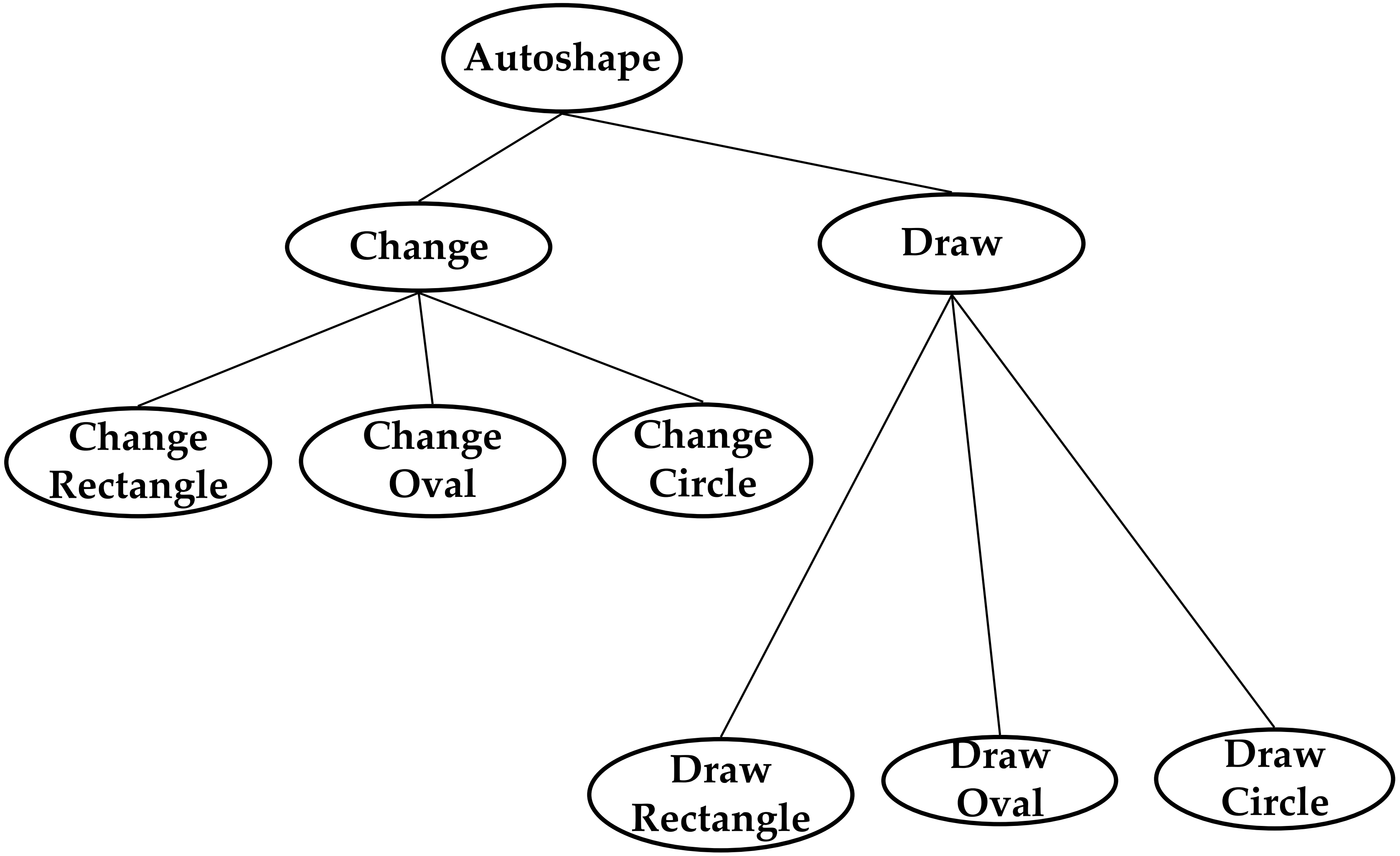
Functional Decomposition

- 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

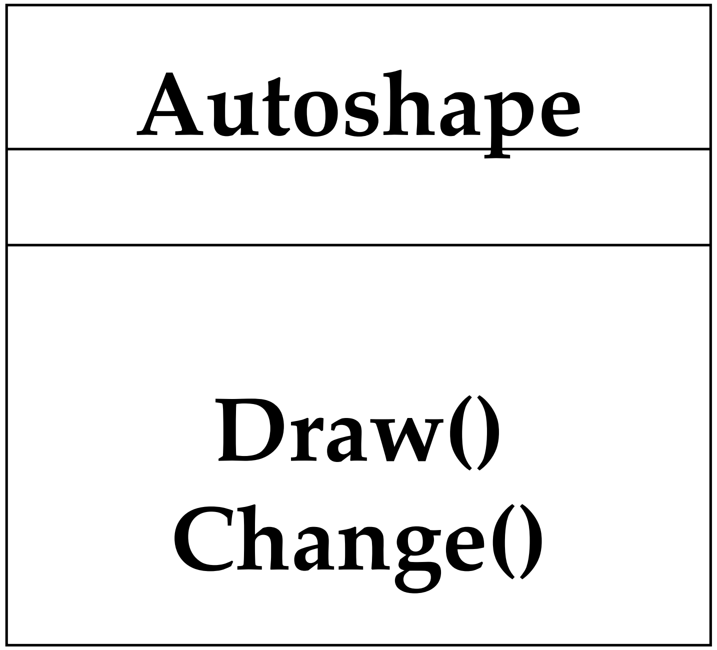
- Example: Microsoft Powerpoint’s Autoshapes
  - How do I change a square into a circle?



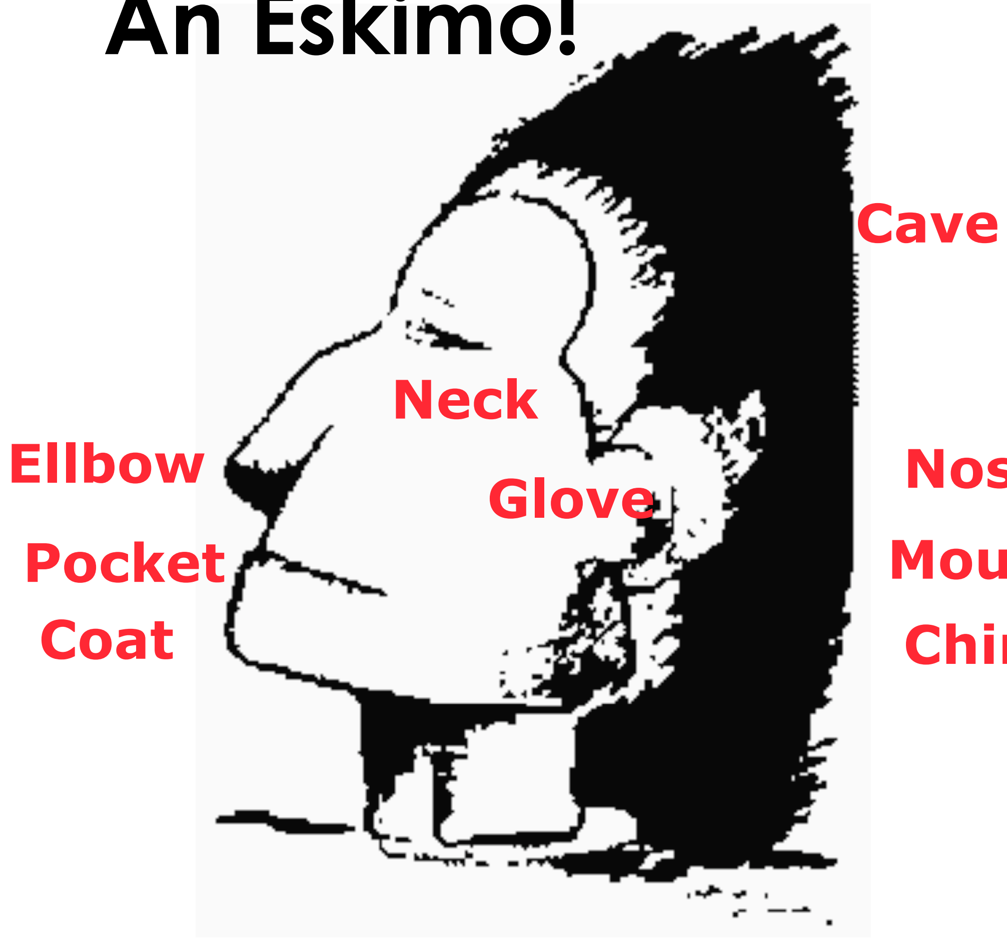
Functional Decomposition: Autoshape



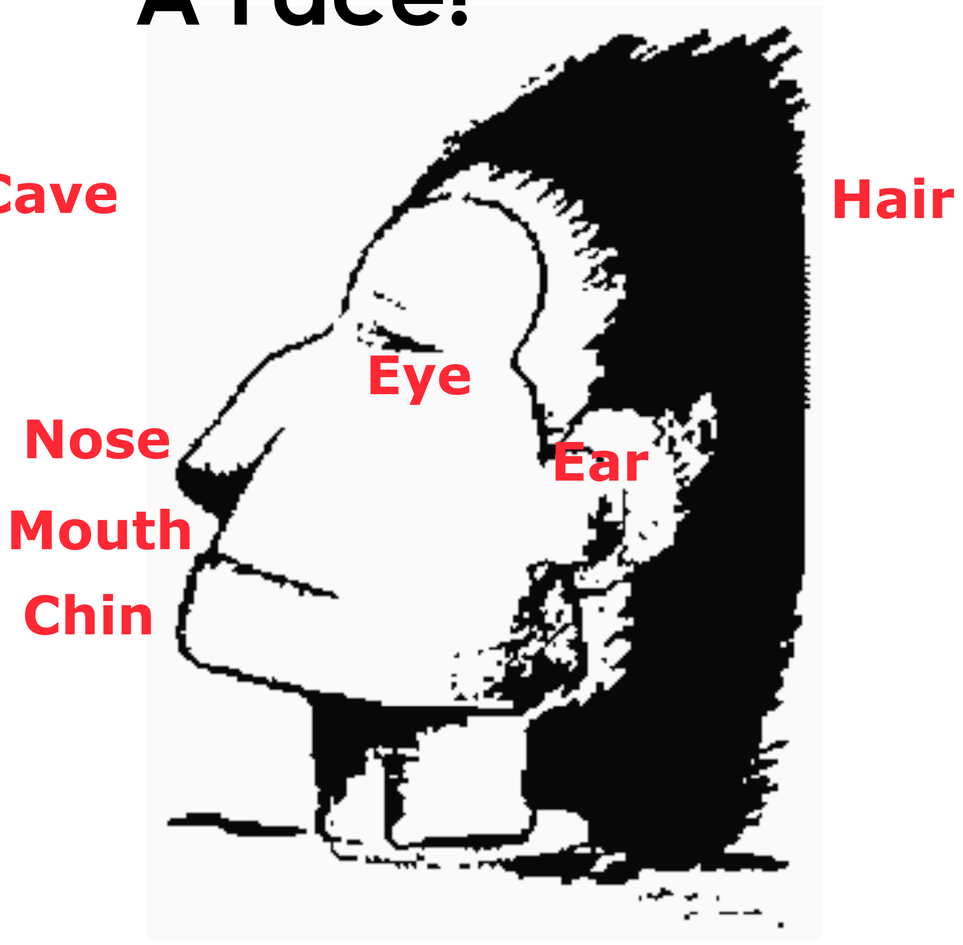
Object-Oriented View



An Eskimo!



A Face!



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**

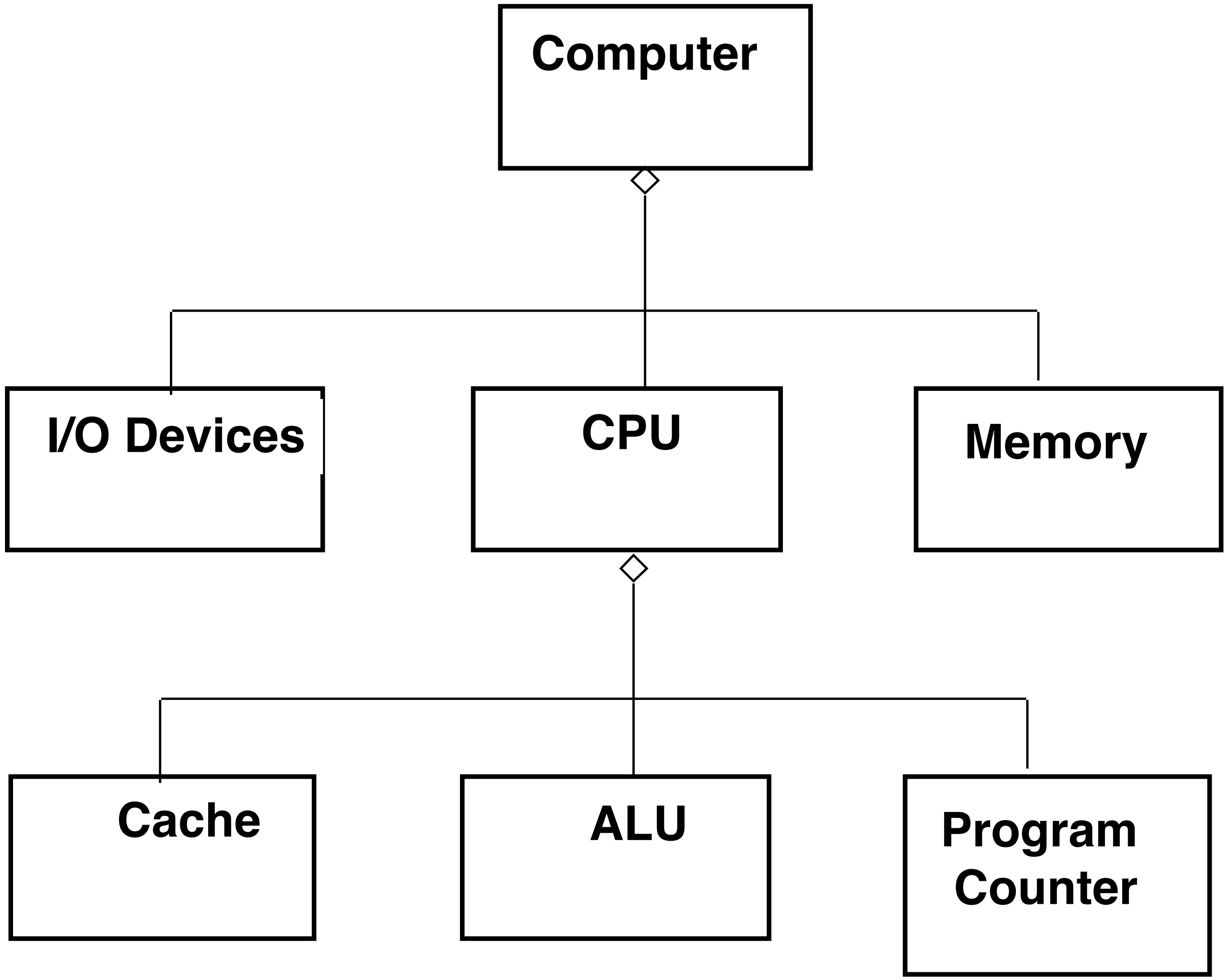
  Identify the purpose of a system

3. Hierarchy

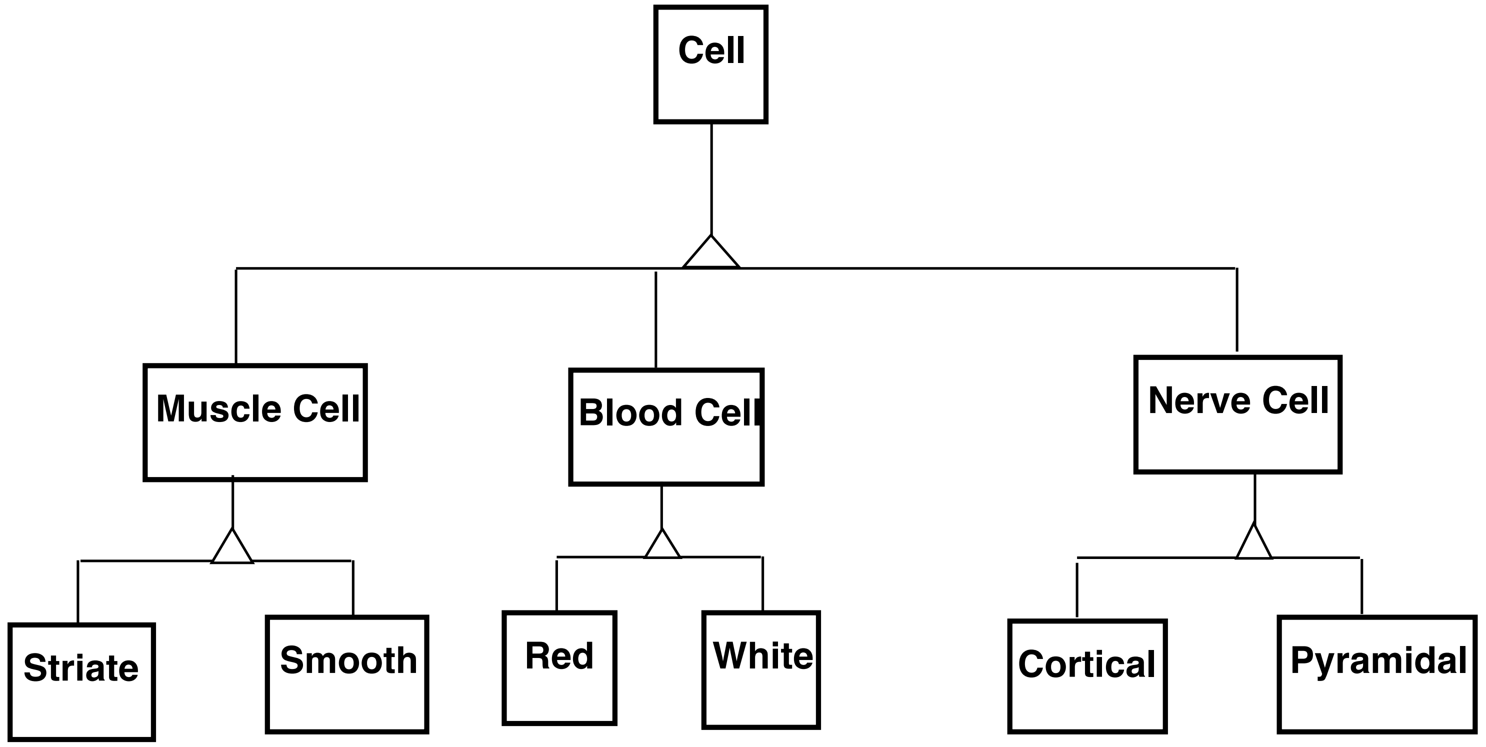
- So far we got abstractions
  - This leads us to classes and objects
  - “Chunks”

- 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)



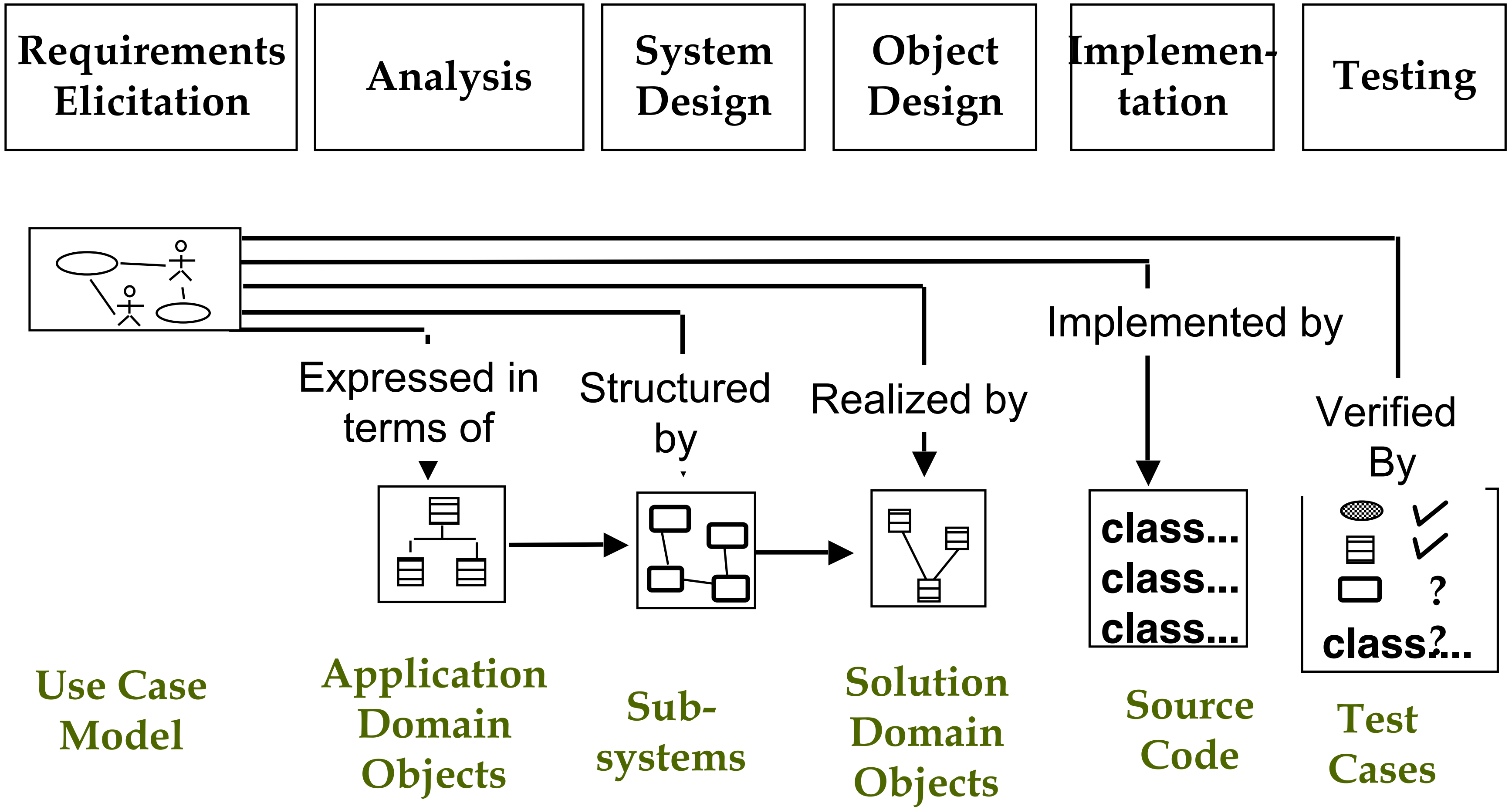
Is-Kind-of Hierarchy (Taxonomy)



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

Software Lifecycle Activities...and their models



Software Lifecycle Definition

- Software lifecycle:
  - Set of activities and their dependency relationships to each other to support the development of a software system

Software Lifecycle Definition (cont'd)

- Typical Lifecycle questions:
  - Which activities should I select for the software project?
  - What are the dependencies between activities?
  - How should I schedule the activities?
- These are the topics of the lecture on software lifecycle modeling

What to do next?

- Read the ARENA case study (Chapter 4.6 in the book)
- Read Chapter 2 for the next lecture on UML Modeling

Summary

- Software development: Problem solving activity
- Goal of software engineering
  - Provide techniques, tools and methodologies
  - Develop quality software for a complex problem within a limited time while things are changing
- Models
  - System models, issue models, task models
- Ways to deal with complexity
  - Decomposition, abstraction, hierarchy
  - Functional & object-oriented decomposition
- Ways to do deal with change
  - Software lifecycle
  - Configuration management, Rationale management, Project management