Software Engineering SS 2006

Lecture 1: Introduction

Prof. Bernd Bruegge, Ph.D.

Applied Software Engineering

Technische Universitaet Muenchen

Intended audience

- Bachelor in Informatics
- Master in Informatics
- Bachelor in Information Systems
- Master in Information Systems
- Master in Applied Informatics
- Master in computational science and engineering (CSE)
- Students taking Informatics as a minor ("Nebenfach").

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
    - Preferably object-oriented such as Java or C++
  - 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 and Locations

- Main lecture: HS 1, 00.02.001
  - Tuesdays 12:45 - 13:30
  - Wednesdays 8:30 - 10:00
- Exercises:
  - Registration starts today at 15:00
  - Registration ends Friday, April 20th 12:00
  - Exercise sessions start on Monday, April 23th
- Written Exams:
  - Mid-term: 2 June 2007, 13:00-15:00
  - Final: 21 July 2007, 13:00-15:00

Grading Criteria

The final grade is the weighted average of the mid term (30%) and final grades (70%)

- To pass this course your final grade must be 4.0 or better
- Successful participation in the exercises is an admission requirement for the final exam
  - If you participation is excellent, you can get a bonus of 1/3 on the final grade (e.g., this can get you from 2.3 to 2.0)
  - The bonus applies only if your grade is 4.0 or better
- Information on the participation is available on the exercise portal
  - http://wwwbruegge.in.tum.de/twiki/bin/view/Lehrstuhl/SoftwareTechnikSoSe2007Exercises
- Hours per week: 3 hours (lecture) + 1 hour (exercises)
- ECTS Credits: 5.0.

Acquire Technical Knowledge

- Understand system modeling
- Learn about modeling notations (Unified Modeling Language UML, Object Constraint Language OCL)
- Learn about different modeling methods
- Learn how to use tools
- Become proficient in testing
- Become proficient in model-based software development.

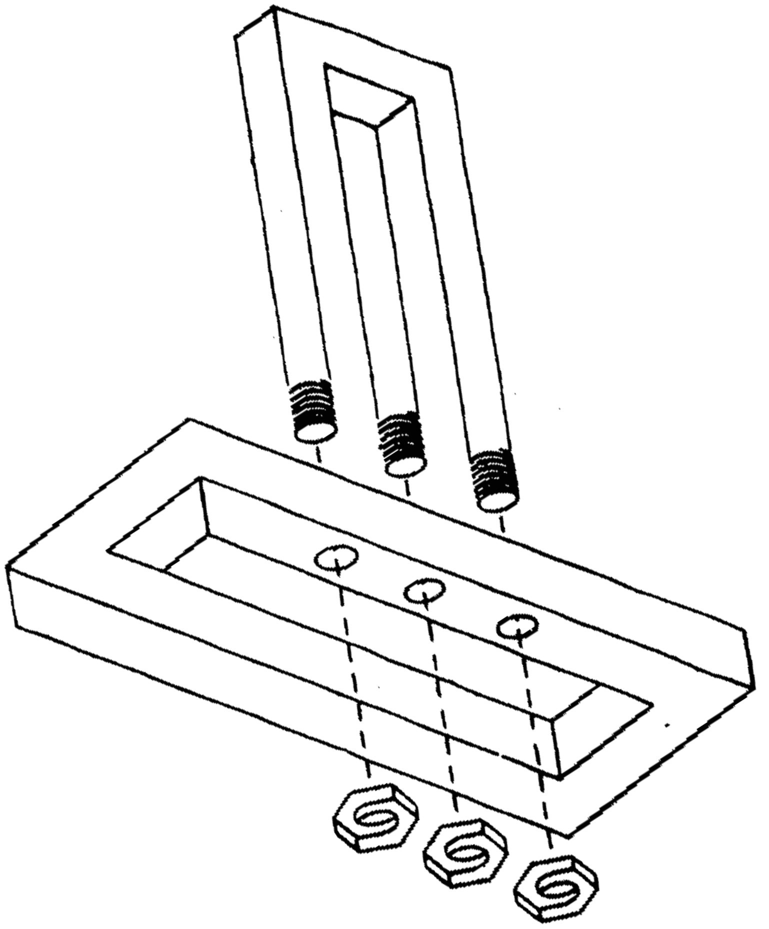
Acquire Managerial Knowledge

- Learn the basics of software project management
- Understand how to manage with a software lifecycle
- Be able to capture software development knowledge (Rationale Management)
- Manage change: Configuration Management
- Learn the basic methodologies
  - Traditional software development
  - Agile methods.

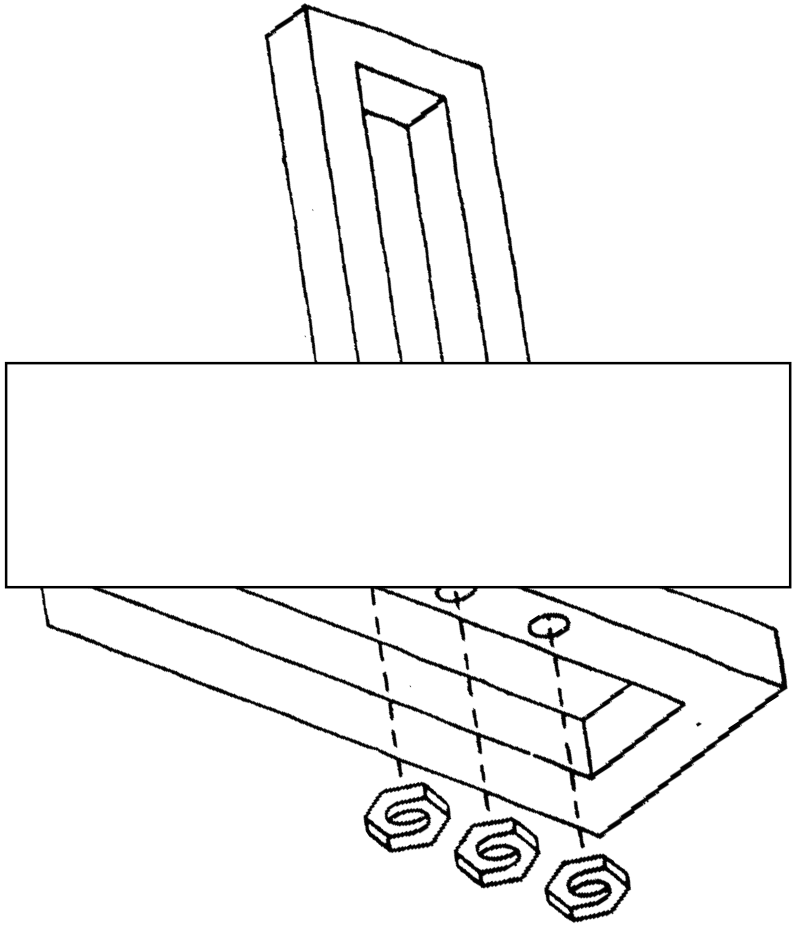
Outline of Today's Lecture

- Modeling complex systems
- Dealing with change
- Concepts
  - Abstraction
  - Modeling
  - Hierarchy
- Organizational issues
  - Lecture schedule
  - Exercise schedule
  - Associated Project

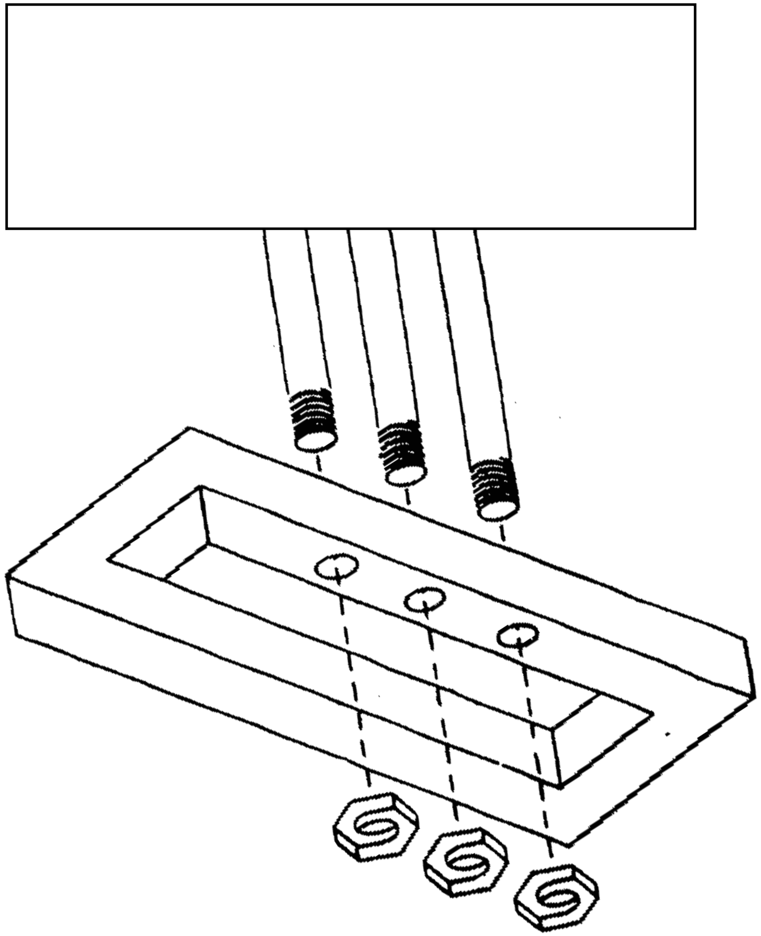
Can you develop this system?



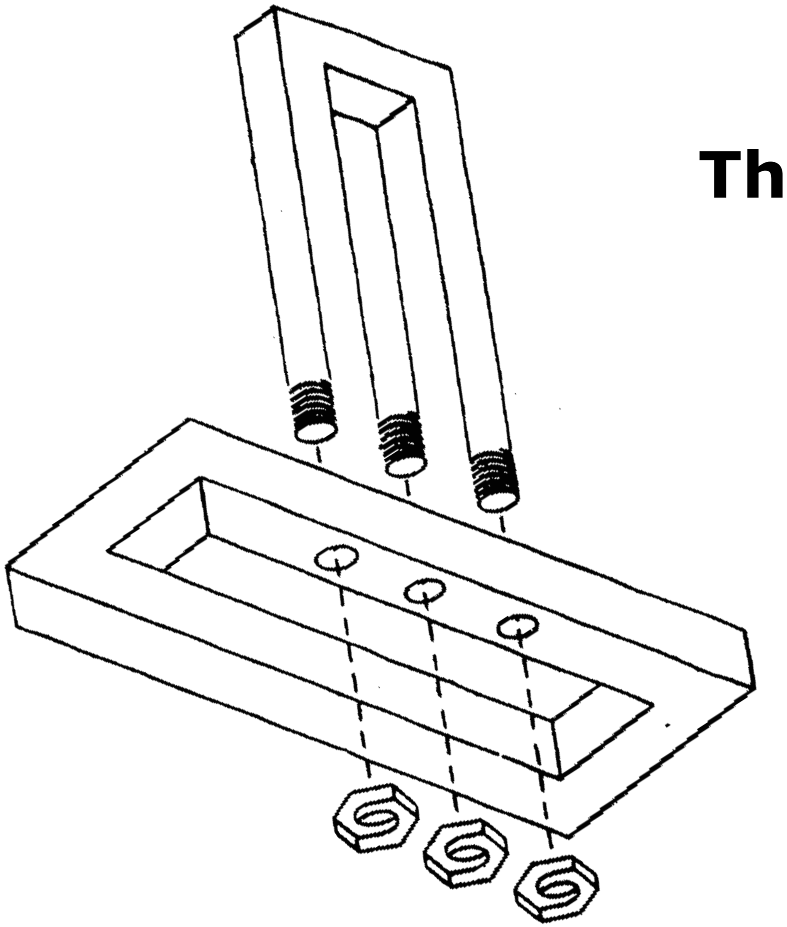
Can you develop this system?



Can you develop this system?

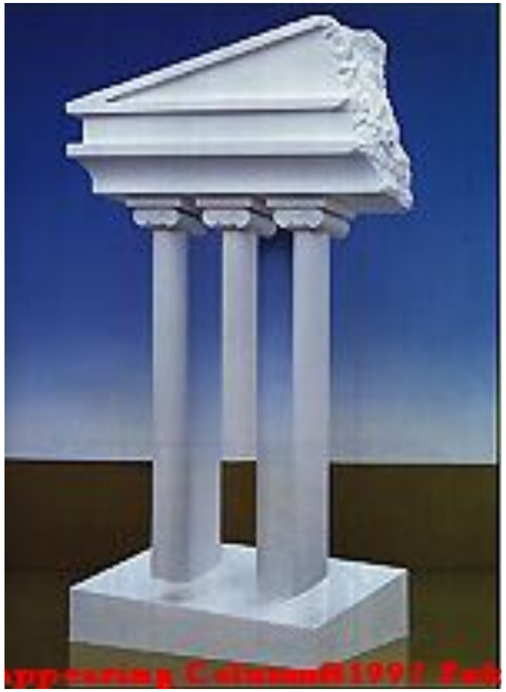


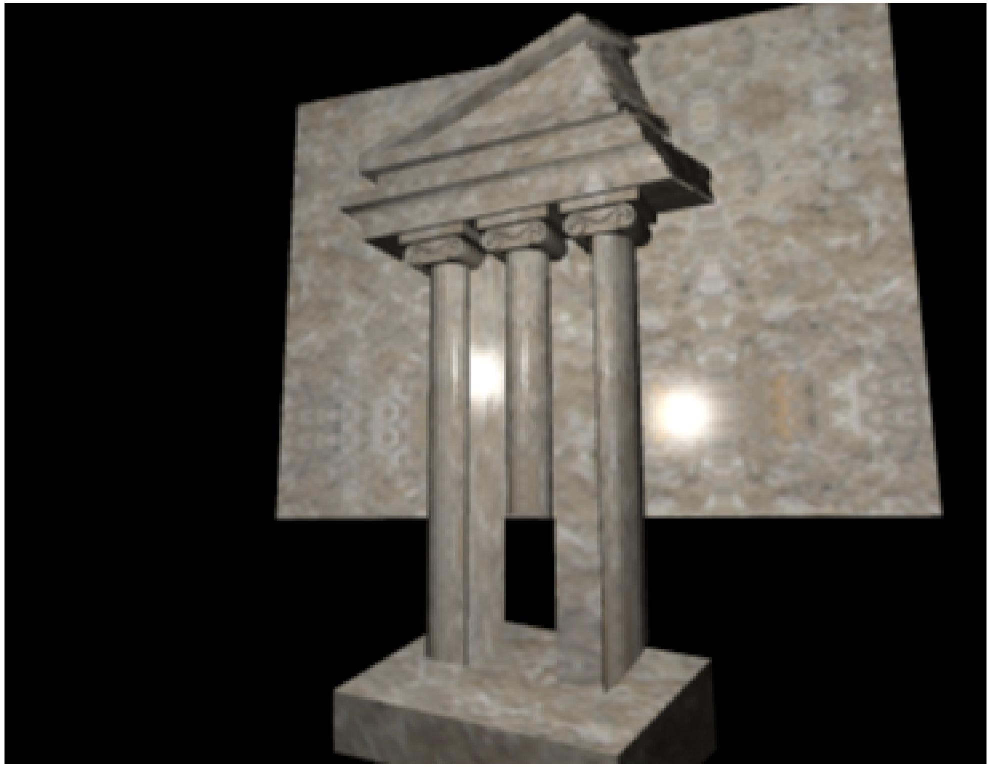
Can you develop this system?



The impossible Fork

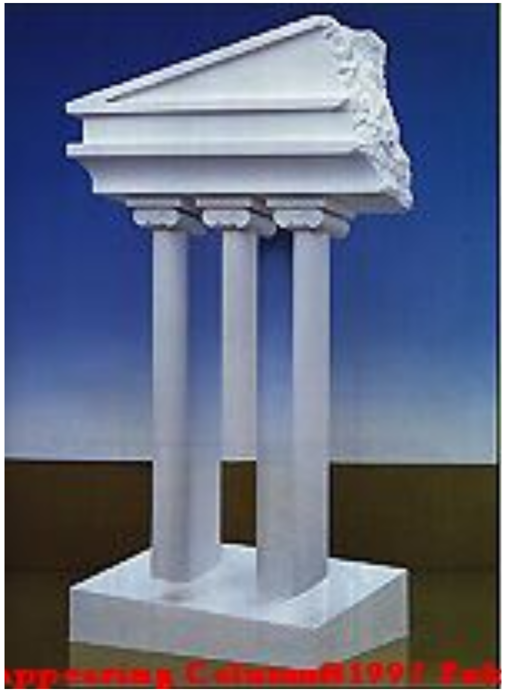
Physical Model of the impossible Fork (Shigeo Fukuda)

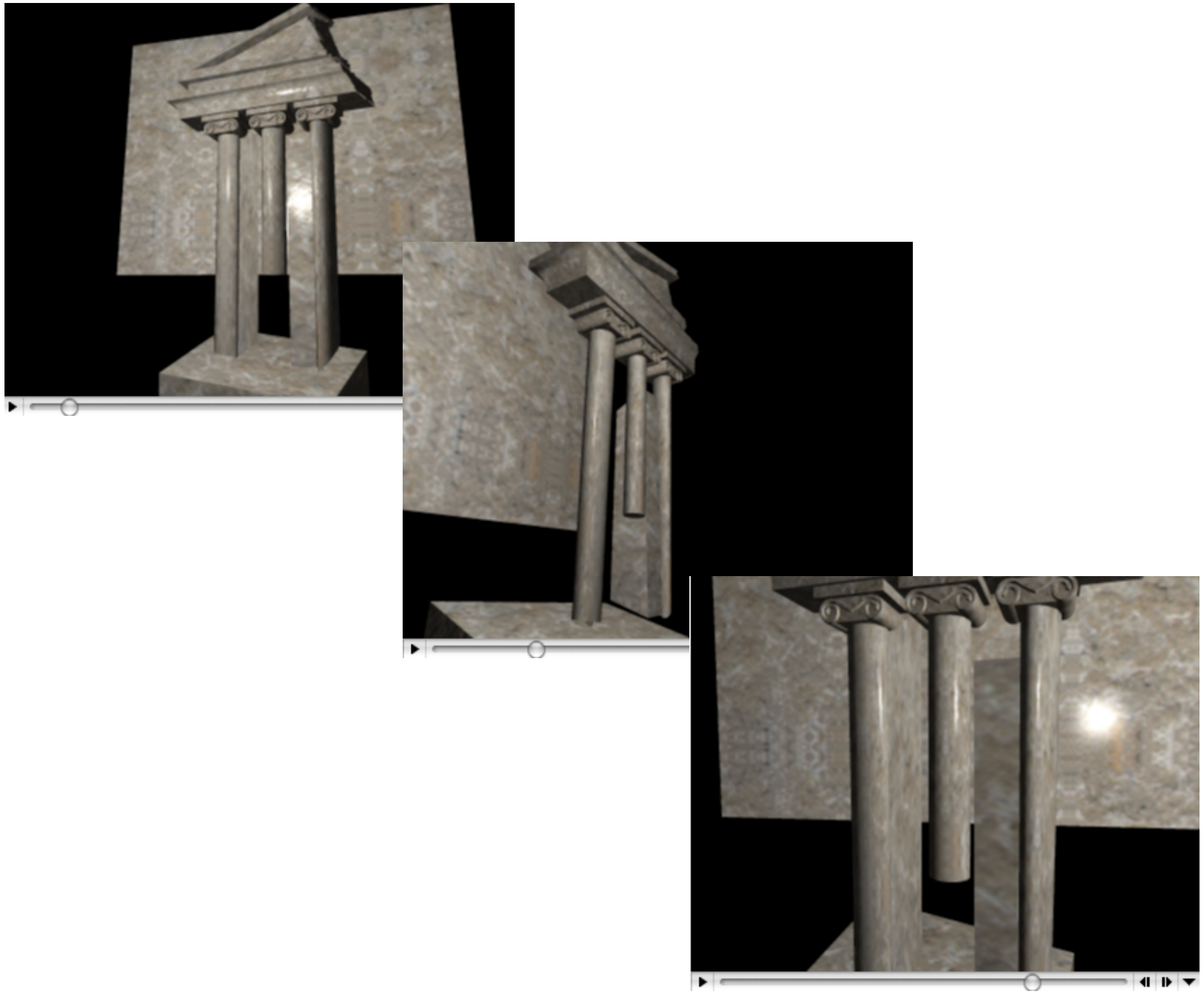




From: http://illusionworks.com/mod/movies/fukuda/DisappearingColumn.mov

Physical Model of the impossible Fork (Shigeo Fukuda)



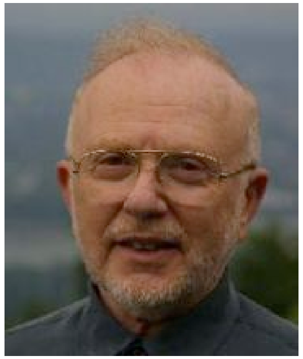


From: http://illusionworks.com/mod/movies/fukuda/DisappearingColumn.mov

Why is software development difficult?

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

David Lorge Parnas is an early pioneer in software engineering who developed the concepts of modularity and information hiding in systems which are the foundation of object oriented methodologies.



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. Engineering

- Computer Scientist
  - Assumes techniques and tools have to be developed.
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for a problem formulated by a client
  - Uses computers & languages, techniques and tools
- Software Engineer
  - Works in multiple application domains
  - Has only 3 months...
  - …while changes occurs in the problem formulation (requirements) and also in the 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 developed with a given *budget* before a given *deadline* while *change* occurs

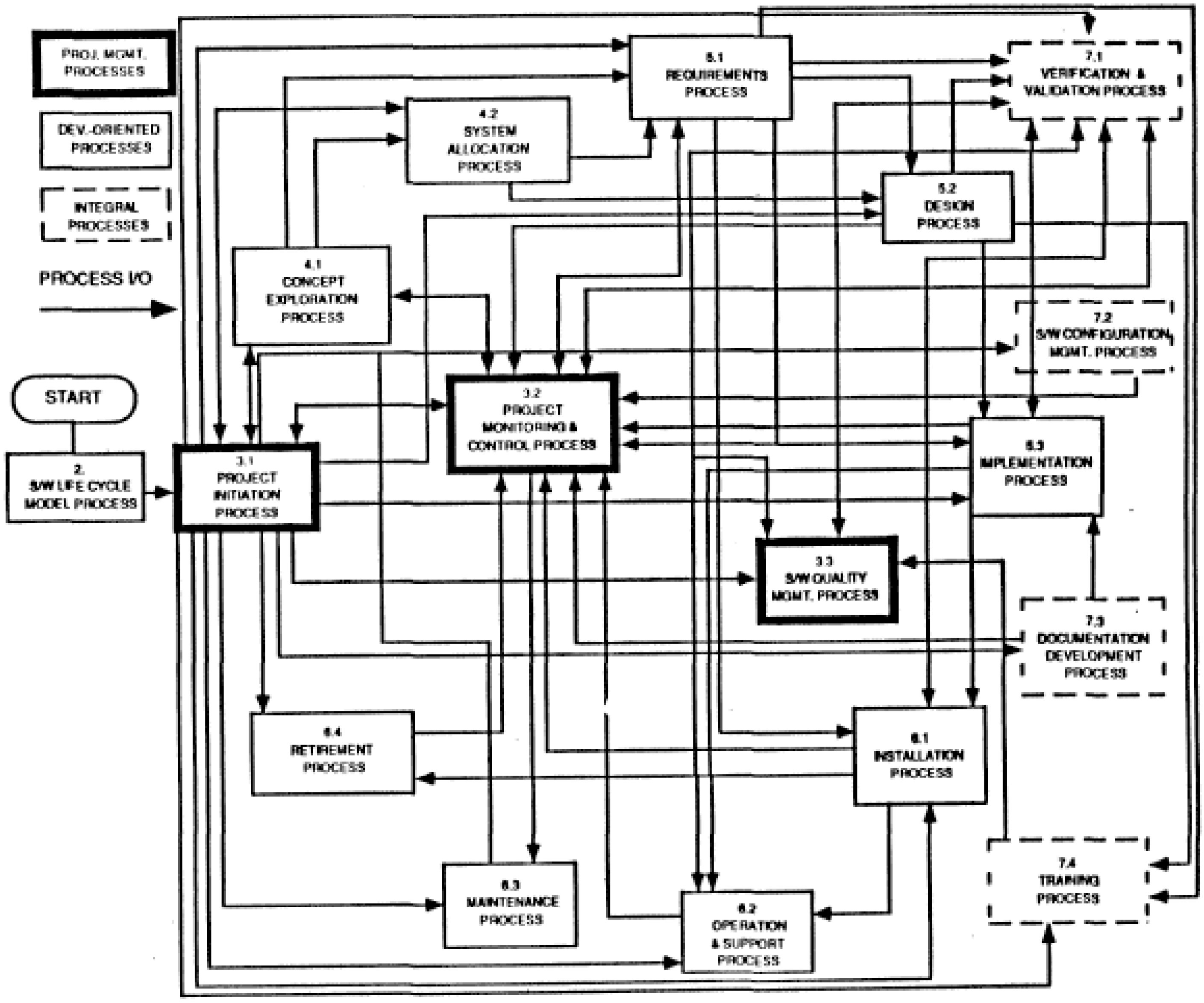
Challenge: Dealing with complexity and change

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

You want to avoid 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

Course Outline

Dealing with Complexity

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

Dealing with Change

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

Application of these Concepts in the Exercises.

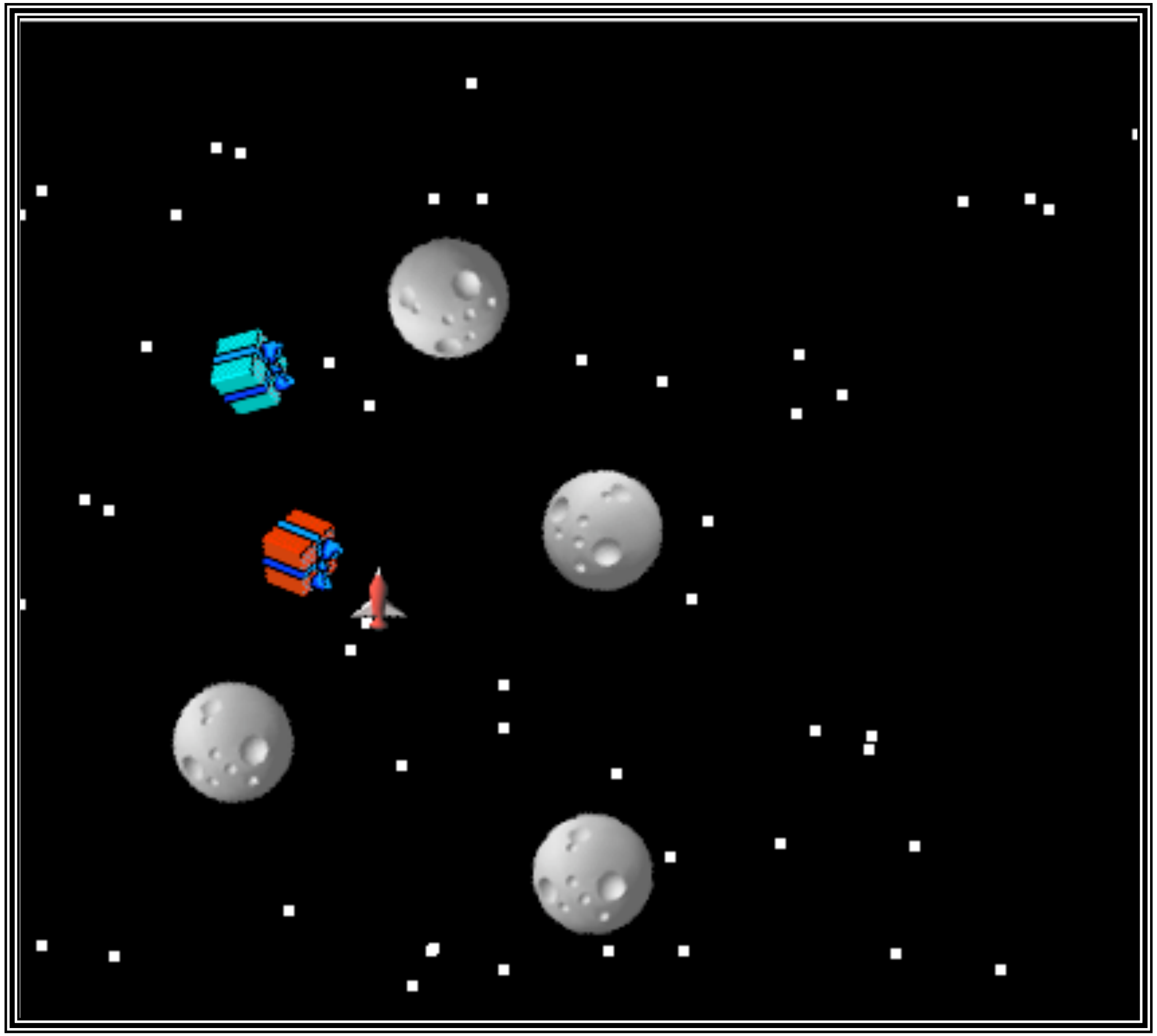
Lecture Schedule

April 17	Introduction	April 18	Introduction to the UML Notation
April 24	Advanced concepts in UML	April 25	Requirements Elicitation
May 1	System Modeling I	May 2	System Modeling II
May 8	System Design I	May 9	System Design II
May 15	Object Design: Reuse	May 16	Intro: Design Patterns
May 22	Design Patterns I	May 23	Design Patterns II
May 29	----- (Holiday)	May 30	Software Architecture
June 05	Object Design: Specification	June 06	OCL (Object Constraint Language)
June 12	*Change Management*	June 13	Mapping models to code
June 19	Unit Testing	June 20	Integration Testing
June 26	System Testing	June 27	Lifecycle Modeling
July 3	*Risk Management*	July 4	Examples of Lifecycle Models
July 10	Agile Methodologies I	July 11	Agile Methodologies II
July 17	Rationale Management	July 18	Putting it all together

Exercises

- The exercises will include a project based on existing systems called Arena and Asteroids
  - Arena is a game management system
  - Asteroids is a specific game
- Both of these systems will also be used in the lectures to illustrate and apply software engineering concepts
  - We will actually include one exercise into a lecture
  - Details will be announced ahead of time
- Project specific models, documents and source code will be made available incrementally during the exercises.

Asteroids



Exercises

- The exercises are organized in groups coached by tutors
- Each group has one exercise session (1 hour) per week
- Registration, attendance in the exercise sessions and accomplishment of the homework are mandatory to pass the lecture.

Exercise - Registration

- There are 19 time-slots with a limit of 20 participants each.
- Registration online:
  https://grundstudium.informatik.tu-muenchen.de/anmeldung
- For the exercise registration you need a certificate. See
  - http://ca.informatik.tu-muenchen.de/userca/
- Registration starts today at 15:00
- Registion closes on Friday, April 20th at 12:00
- Exercises start on Monday, April 24th
  - The starting times vary for the individual groups.

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.

More Questions?

- e
- v

©