SE464/CS446/ECE452
Software Design and Architecture

Instructor:

Krzysztof Czarnecki

Outline for today

➔Introduction
- Course description
- Software design and architecture basics

About my background

- Moved from industry to academia in 2003
- Worked for the Research and Technology corporate division of DaimlerChrysler AG in Germany for 8 years
- Expertise in object technology, software reuse, generative and model-driven software development
- Research, consulting and development projects in IT & embedded control software and development tools

Some of my past customers

Mercedes-Benz
Passenger Vehicles

Mercedes-Benz
Commercial Vehicles

EADS Military Aricraft

Astrium Space

Other: Debis Systemhaus, AEG Postal Automation, MTU Aero Engines

Outline for today

- Introduction

➔Course description

- Software design and architecture basics

Course components

- 3 lectures
  - Mondays, Wednesdays, and Fridays
- 1 tutorial
- 1 big project

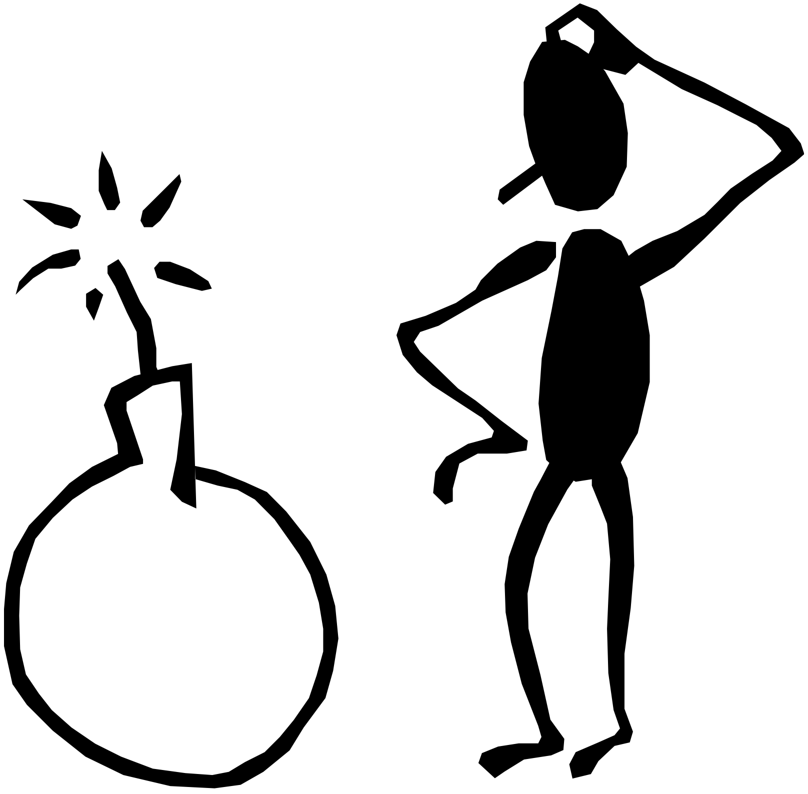
Course website

- lecture and tutorial schedule
- lecture slides and additional materials
- recommended books
  - Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns -- Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
  - Mary Shaw and David Garlan. Software Architecture -- Perspectives on an Emerging Discipline. Prentice Hall, 1996
  - …
- project description
- grade allocation
- course news system

- cheating policy: you cheat you fail

Big project component

- Design and implement software for the IP Phone system specified in the requirements course
- Groups of 4
- A major task...
  - Your capstone project, not just a class project
  - *Frontloaded*: Major portion due before midterms
- 50% of your grade
- Start working on it from week 1!
  - Go to the tutorial
  - Checkout the lab



Goals of this course

- Familiarize with concepts and methods of software design and architecture
- Learn how to perform architectural design and OO design and basic project management tasks using examples
- Experience design and architecture in a larger project
  - Not all lecture material covered by the project

➔ Note (by words of Richard Taylor):

- “Scratching the surface of software engineering”
- “Fitting you to become an amateur software engineer”

Course outline

- Introduction to design
- Software lifecycle and process models; XP
- Introduction to architecture
  - Basic design principles (modularity, coupling & cohesion, interfaces)
  - Documenting architectures
- OO analysis & design
- OO design patterns and refactoring
- Architectural styles & patterns
- Enterprise patterns
- Embedded software patterns
- Project discussion
- Product-line architectures and OO frameworks
- Model driven development

Outline for today

- Introduction
- Course description

➔Software design and architecture basics

What Is Design?

- Requirements specification was about the WHAT the system will do
- Design is about the HOW the system will perform its functions
  - provides the overall decomposition of the system
  - allows to split the work among a team of developers
  - also lays down the groundwork for achieving non-functional requirements (performance, maintainability, reusability, etc.)
  - takes target technology into account (e.g., kind of middleware, database design, etc.)

Software Development Activities

- Requirements Elicitation
- Requirements Analysis (e.g., Structured Analysis, OO Analysis)
  - analyzing requirements and working towards a *conceptual* model *without* taking the target implementation technology into account
  - useful if the conceptual gap between requirements and implementation is large
  - part of requirements engineering (but may produce more than what is going to be part of the requirement spec)
- Design
  - coming up with solution models *taking* the target implementation technology into account
- Implementation
- Test
- ...

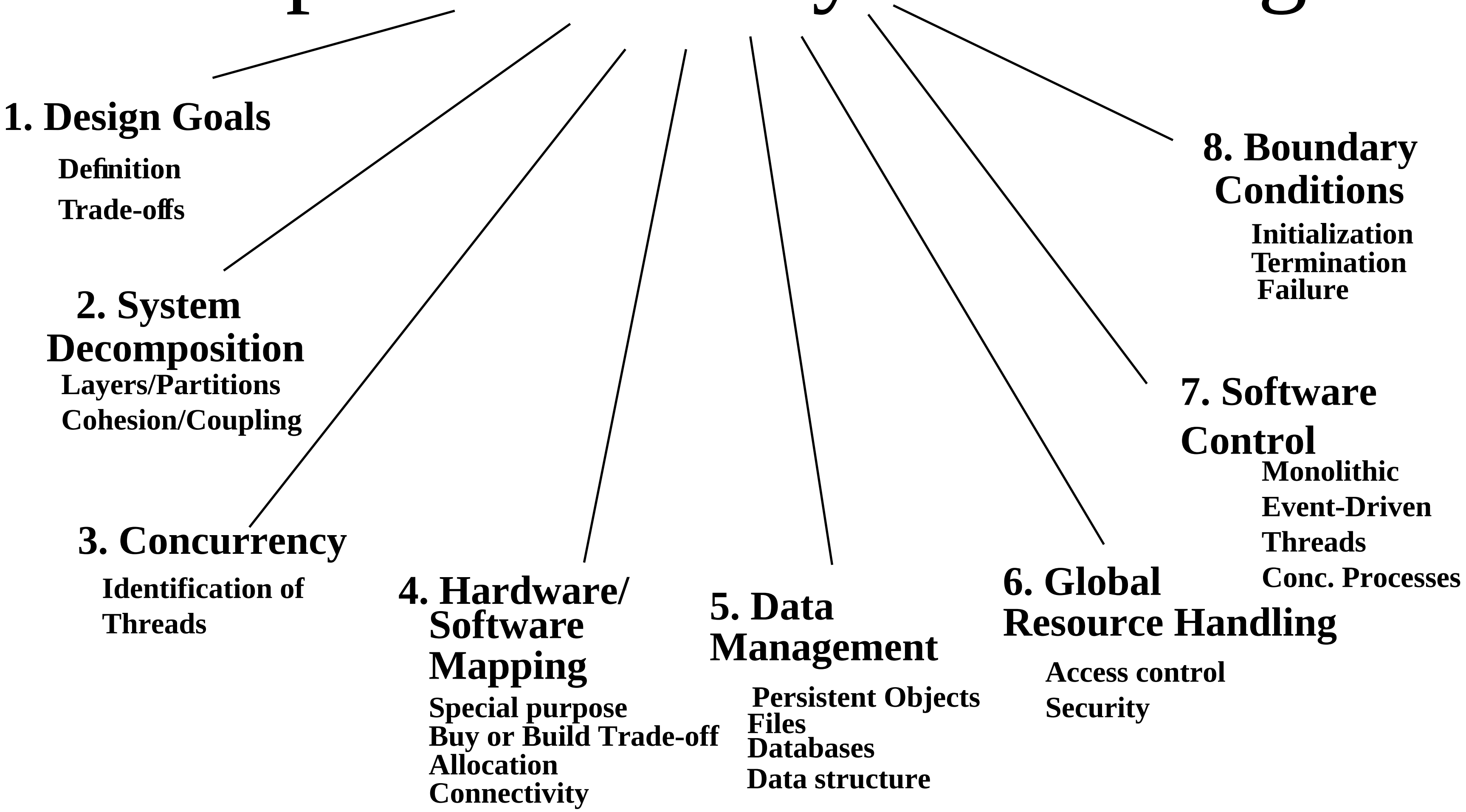
Levels of Design

- Architectural design (also: high-level design)
  - architecture - the overall structure: main modules and their connections
  - design that covers the main use-cases of the system
  - addresses the main non-functional requirements (e.g., throughput, reliability)
  - hard to change
- Detailed design (also: low-level design)
  - the inner structure of the main modules
  - may take the target programming language into account
  - detailed enough to be implemented in the programming language

The Design Process

- Study and understand the problem from different viewpoints
- Identify potential solutions and evaluate the tradeoffs
  - Design experience, reusable artifacts, simplicity of solutions
  - Sub-optimal, but familiar solutions often preferred – advantages/disadvantages well known
  - Design is about making tradeoffs!
- Develop different models of system at different levels of abstraction and for different perspectives

Complexities of System Design



1. Design Goals

Definition

Trade-offs

2. System Decomposition

Layers/Partitions

Cohesion/Coupling

3. Concurrency

Identification of Threads

4. Hardware/ Software Mapping

Special purpose

Buy or Build Trade-off

Allocation

Connectivity

5. Data Management

Persistent Objects

Files

Databases

Data structure

6. Global Resource Handling

Access control

Security

7. Software Control

Monolithic

Event-Driven

Threads

Conc. Processes

8. Boundary Conditions

Initialization

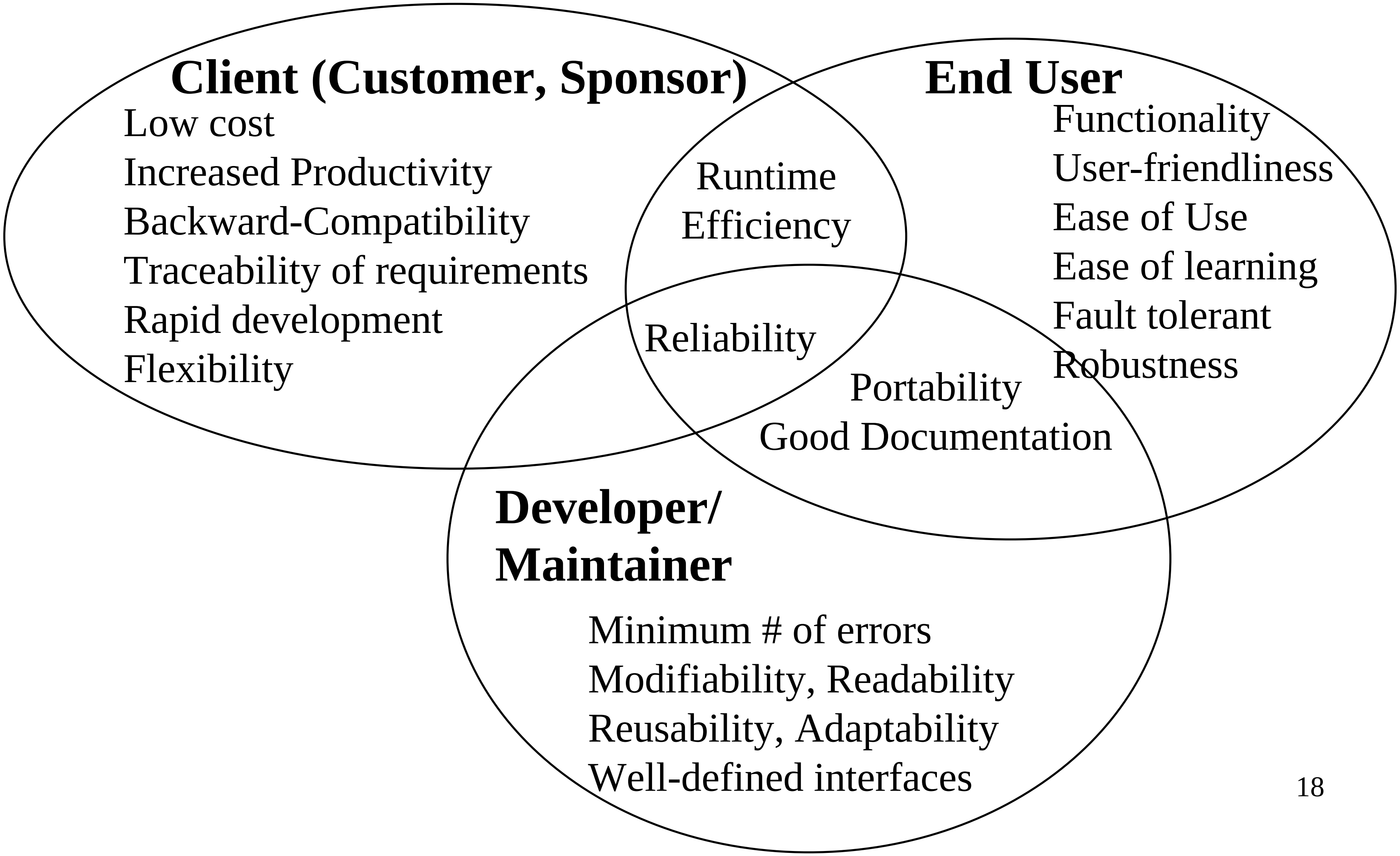
Termination

Failure

List of Design Goals

- Reliability
- Modifiability
- Maintainability
- Understandability
- Adaptability
- Reusability
- Efficiency
- Portability
- Traceability of requirements
- Fault tolerance
- Backward-compatibility
- Cost-effectiveness
- Robustness
- High-performance
- Good documentation
- Well-defined interfaces
- User-friendliness
- Reuse of components
- Rapid development
- Minimum # of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility
- …

Relationship Between Design Goals



Client (Customer, Sponsor)

Low cost
Increased Productivity
Backward-Compatibility
Traceability of requirements
Rapid development
Flexibility

End User

Functionality
User-friendliness
Ease of Use
Ease of learning
Fault tolerant
Robustness

Runtime Efficiency

Reliability

Portability
Good Documentation

Developer/ Maintainer

Minimum # of errors
Modifiability, Readability
Reusability, Adaptability
Well-defined interfaces

Typical Design Trade-offs

- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Rapid development vs. Functionality
- Cost vs. Reusability
- Backward Compatibility vs. Readability

Challenges in Design

- Complexity
  - Often arbitrary, dependent on designer rather than problem (“accidental complexity”)
- Conformity
  - Often expected to conform to other software (e.g., legacy, standards)
- Changeability
  - Needs to support change due changing requirements, constraints, etc.
- Invisibility
  - No visible link from between design plans and product

The Design Process

- Cannot mechanically produce a design
- Design requires intelligence
- Design requires experience
- The intelligence can be guided by design methods and techniques, but it can not be replaced.

Top-Down vs. Bottom-Up Design

- Top-Down
  - Recursively partition problem into smaller sub-problems
  - Continue until tractable solutions found
  - Note: Not practical for large system in its pure form
- Bottom-Up
  - Assemble, adapt, and extend existing solutions to fit the problem
- In practice: A combination of both
  - Decompose large problems into smaller, but using previous design knowledge
  - Use existing components and solutions
  - Perhaps tackle problematic portions first

Design Methods

- Design methods provide guidance
- Different flavors
  - Heavyweight methods
    - Highly structured and documentation oriented methods
    - Usually generate mega amounts of graphical documentation
  - Agile methods
    - “Travel light”
  - Agile model-based methods
    - Best of both worlds
    - Still in early development

Design Methods

- Action oriented approach
  - e.g., data-flow design
  - favors the functional view
  - appropriate if actions are the main aspects of a system
- Data oriented approach
  - e.g., Jackson’s design method
  - favors the data view
  - appropriate if data are the main aspects of a system
- OO approach
  - looks at both actions and data at the same time
  - system viewed as a collection of objects not functions
  - system state is decentralized – each object manages its own state information
  - objects have attributes defining state and operations which act on attributes
  - conceptually, objects communicate via messages
- Domain-specific approach
  - A set of modeling views and concepts specifically developed for a class of problems

Design Notations

- Taking an abstraction implies that you are making a decision about which details are important and which can be ignored. This decision is based on your viewpoint.
- Four key viewpoints in software design:
  - *Structural* - the static properties of the software
  - *Behavioral* - cause and effect;
  - *Functional* - what tasks the software performs
  - *Data modeling* - the data objects used

This Week…

- Tutorials this week
  - – Project Introduction