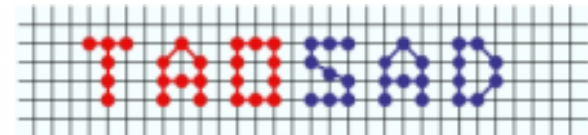


Classifying Software Architecture Design Methods



Bedir Tekinerdoğan

University of Twente
Department of Computer Science
Enschede, The Netherlands
e:mail – bedir@cs.utwente.nl
<http://www.cs.utwente.nl/~bedir/>



<http://trese.cs.utwente.nl/taosad/>

Software Architecture is ...

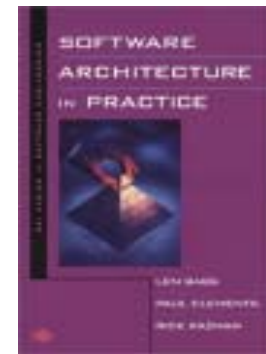
- The gross-level structure of the system
- which comprise software components,
- and the relationships among them.



*How to produce a
software architecture?*

Architecture Design Methods

- The Unified Software Development Process, G. Booch, J. Rumbaugh & I, Jacobson.
- A System of Patterns: Pattern-Oriented Software Architecture, F. Buschman et al.
- (Object-Oriented Modeling and Design, J. Rumbaugh
- Design and use of software architectures, J. Bosch.
- Software Architecture in Practice, P. Clements, L. Bass & R. Kazman
- Applied Software Architecture, C. Hofmeister et al.
- Software Product Lines, P. Clements & L. Northrop
- Software Architectures: Perspectives on an Emerging Discipline, M. Shaw
- Synthesis-Based Software Architecture Design, Tekinerdogan & Aksit
- ...



Which one to select?



B. Tekinerdogan and M. Aksit. Classifying and Evaluating Architecture Design Methods, in Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Aksit (Ed.), Kluwer Academic Publishers, pp. 3 - 27, 2001.

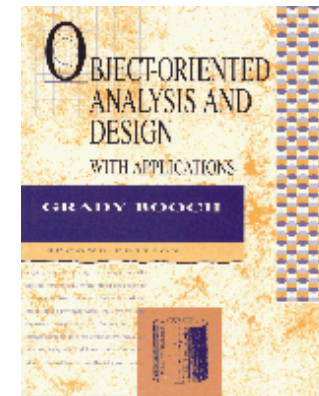
Classification of Methods

- What is the source of architectural abstractions?
 - Artifact-driven
 - Use case/Scenario-driven
 - Pattern-driven
 - *Domain-driven*



Artifact-driven

- Start from textual requirements
- Look at artifact types in the method and try to identify artifacts from requirements specification using heuristic rules.
- Group the related artifacts in *subsystems*, these are the architectural components.
- Define the relations between subsystems.



Example: PC Factory

Requirements Specification: PC FACTORY
Date: 16-October 2003

A software system for a computer company, which consists of two departments, a factory and sales and marketing department. The factory assembles desktop PCs and tower PCs. All the components of a PC are delivered by different external suppliers. A PC consists of one monitor, a cabinet, and a keyboard. The cabinet includes a chassis. A chassis on its turn is composed of a bus, floppy disk drive, an optional CD-ROM drive, a memory unit, CPU, and power supply. A bus may incorporate a network card. A memory unit includes many RAM chips. The sales and marketing department administers properties of each PC, like the type, weight, make, price, amortization, power consumption etc. A client uses a purchase order to order PCs or set of computer components from the company.

**Not precise,
Ambiguous,
Redundant,
Difficult to understand,
Not complete...**



Applying Heuristics

Requirement Specification

Requirements Specification
PC FACTORY
Date: 16-October 2002

A software system for a computer company, which consists of two departments, a factory and sales and marketing department. The factory assembles desktop PCs and tower PCs. All the components of a PC are delivered by different external suppliers. A PC consists of one monitor, a cabinet, and a keyboard. The cabinet includes a chassis. A chassis on its turn is composed of a bus, floppy disk drive, an optional CD-ROM drive, a memory unit, CPU, and power supply. A bus may incorporate a network card. A memory unit includes many RAM chips.

The sales and marketing department administers properties of each PC, like the type, weight, make, price, amortization, power consumption etc. A client uses a purchase order to order PCs or set of computer components from the company.

Heuristic Rules

Tentative Class identification

Extract nouns from the problem statement.

Select nouns as tentative classes.

Extract tentative classes from application domain knowledge.

Extract tentative classes from general knowledge.

Class identification

Class

The identified tentative classes are used to identify classes:

IF tentative class <isRedundant>

THEN eliminate tentative class.

IF tentative class <isIrrelevant>

THEN eliminate tentative class.

IF tentative class <isVague>

THEN eliminate tentative class.

IF tentative class <isAttribute>

THEN select tentative class as Attribute.

IF tentative class <isOperation>

THEN eliminate tentative class.

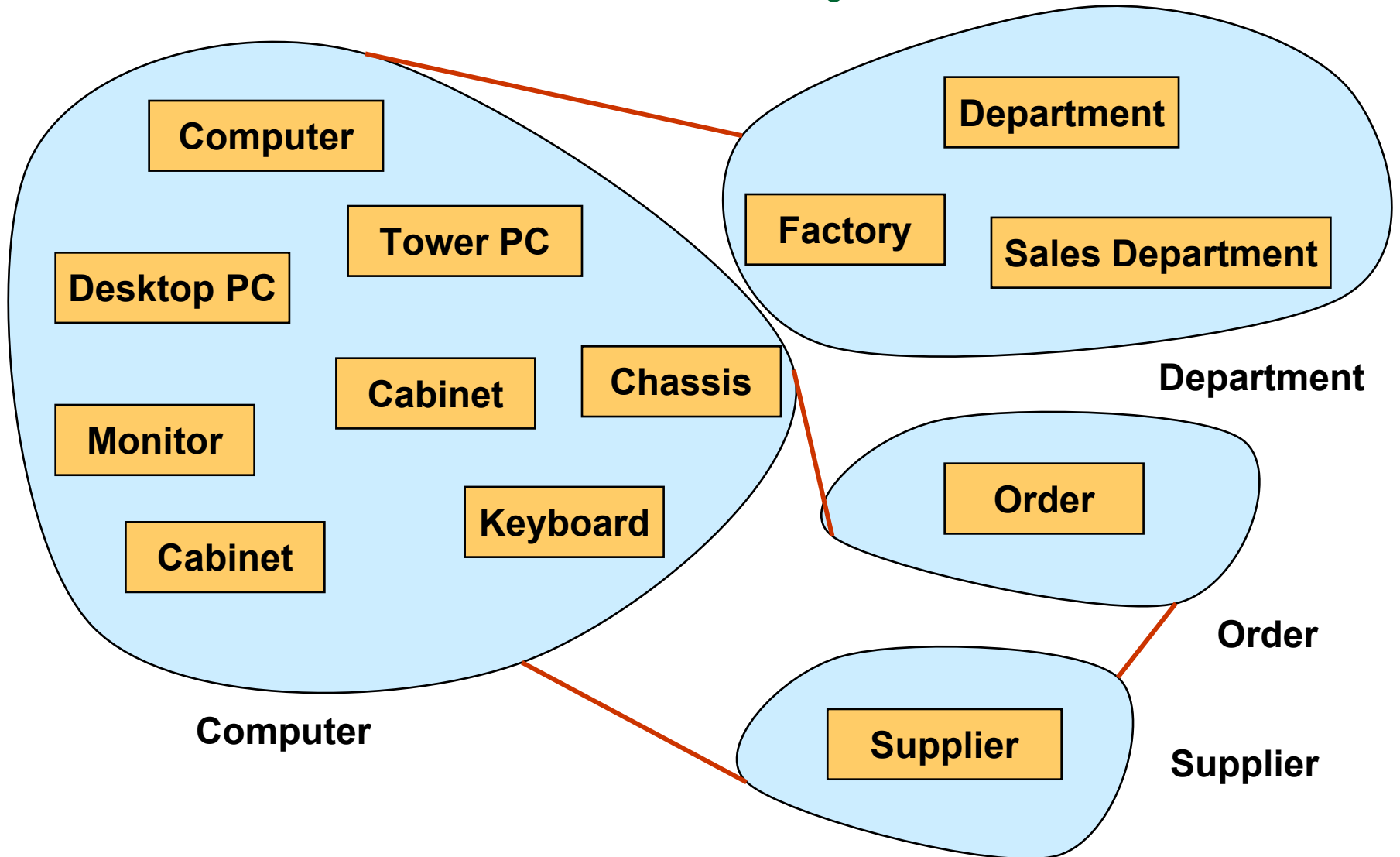
IF tentative class <isRole>

THEN eliminate tentative class.

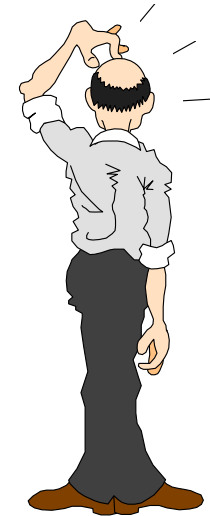
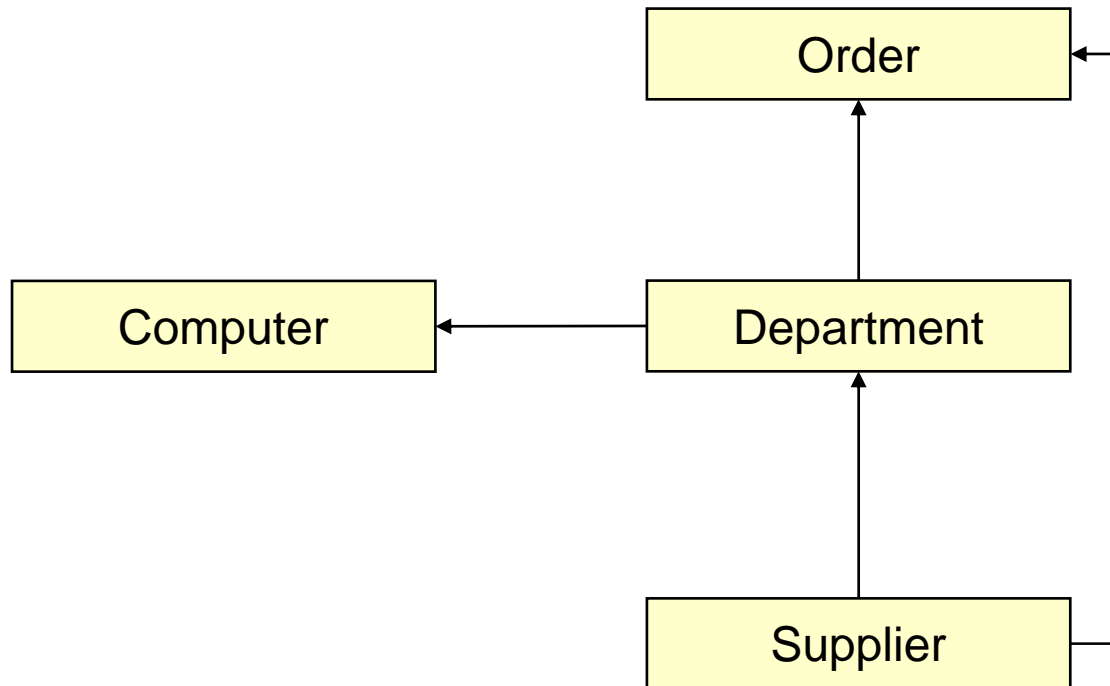
Ok, this is a class, this isn't. This might also be a class, And this one also...



Identified Classes/Subsystems



Identified Architecture



Obstacles Artifact-Driven Approach

- Textual requirements are imprecise and are less useful as a source for deriving architectural abstractions
- Subsystems have poor semantics to serve as architectural components
- Composition of subsystems is not well-supported.

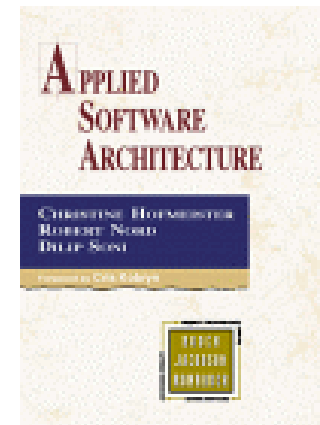


Use case driven

- Extract use cases
- Identify fundamental classes from use cases.
- Group these classes in *packages*, these are the architectural components.
- Define the relations between packages.



<http://www.rational.com/rup/>



Obstacles

- Selecting architecturally relevant use cases is not systematically supported.
- Use cases do not provide a solid basis for architectural abstractions
- Package construct has poor semantics to serve as architectural abstractions.

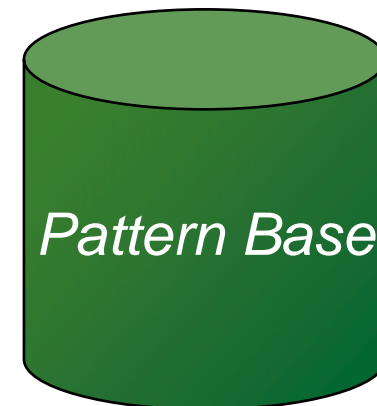
Pattern driven

- Start with requirement specification
- Select appropriate patterns from a pattern base.
- Compose these patterns.



Pattern

- Pattern is a generic and reusable design solution for recurring problems in a given context.
- Each pattern describes a solution, problem and the context.
- Patterns can be used to construct software architectures.
- Examples:
 - Layers, Blackboard, Pipes and Filters, etc.



Obstacles of Pattern-Driven Approaches

- Patterns only might not be sufficient for deriving architectural abstractions.
- Selection of patterns is not well supported and depends on experience of software engineer.
- Applying patterns is not straightforward and requires thorough analysis of the problem.
- Composing patterns is not well supported.

More information...

- B. Tekinerdogan, *Synthesis Based Software Architecture Design*, Ph.D. thesis, University of Twente, The Netherlands, March 2000.
- M. Aksit (Ed.), *Software Architectures and Component Technology: The State of the Art in Research and Practice*, Kluwer Academic Publishers, 2001.
- B. Tekinerdogan and M. Aksit, *Classifying and Evaluating Architecture Design Methods*, in *Software Architectures and Component Technology: The State of the Art in Research and Practice*, M. Aksit (Ed.), Kluwer Academic Publishers, pp. 3 - 27, October 2001.

