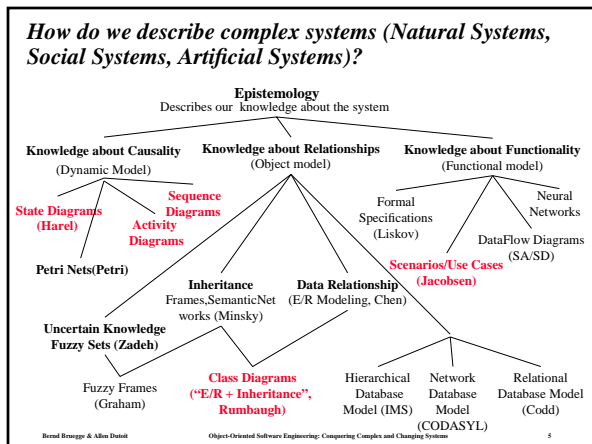
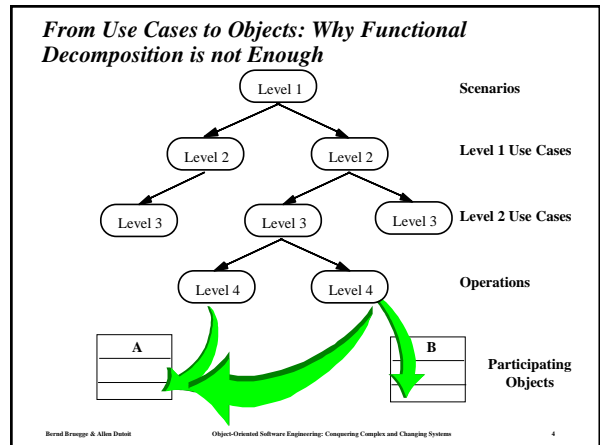
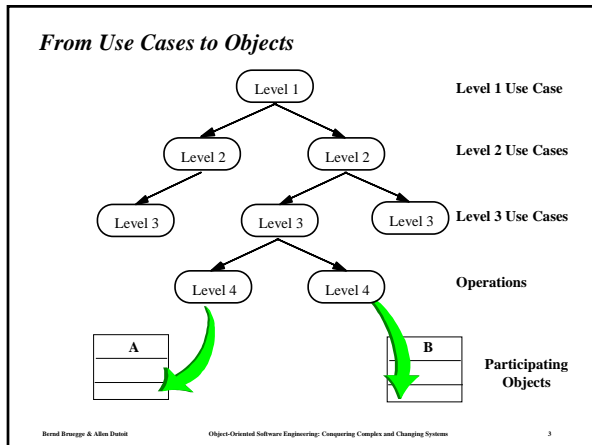


Outline

- ◆ From use cases to objects
- ◆ Object modeling
- ◆ Class vs instance diagrams
- ◆ Attributes
- ◆ Operations and methods
- ◆ Links and associations
- ◆ Examples of associations
 - ◆ Aggregation
 - ◆ Inheritance

Bertel Bruggen & Allen Dutoit Object-Oriented Software Engineering: Conquering Complex and Changing Systems 2



Definition: Object Modeling

- ◆ Main goal: Find the important abstractions
- ◆ What happens if we find the wrong abstractions?
 - ◆ Iterate and correct the model
- ◆ Steps during object modeling
 1. Class identification
 - ◆ Based on the fundamental assumption that we can find abstractions
 2. Find the attributes
 3. Find the methods
 4. Find the associations between classes
- ◆ Order of steps
 - ◆ Goal: get the desired abstractions
 - ◆ Order of steps secondary, only a heuristic
 - ◆ Iteration is important

Bertel Bruggen & Allen Dutoit Object-Oriented Software Engineering: Conquering Complex and Changing Systems 6

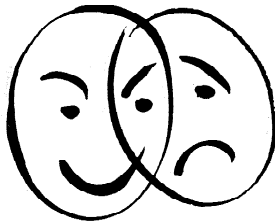
Class Identification

- ◆ Identify the boundaries of the system
- ◆ Identify the important entities in the system
- ◆ Class identification is crucial to object-oriented modeling
- ◆ Basic assumption:
 - ◆ 1. We can find the classes for a new software system (Forward Engineering)
 - ◆ 2. We can identify the classes in an existing system (Reverse Engineering)
- ◆ Why can we do this?
 - ◆ Philosophy, science, experimental evidence

Class identification is an ancient problem

- ◆ Objects are not just found by taking a picture of a scene or domain
- ◆ The application domain has to be analyzed.
- ◆ Depending on the purpose of the system different objects might be found
 - ◆ How can we identify the purpose of a system?
 - ◆ Scenarios and use cases
- ◆ Another important problem: Define system boundary.
 - ◆ What object is inside, what object is outside?

What is This?



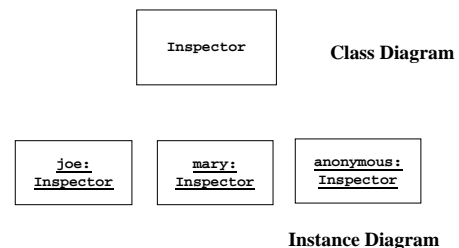
Pieces of an Object Model

- ◆ Classes
- ◆ Associations (Relations)
 - ◆ Part-of-Hierarchy (Aggregation)
 - ◆ Kind-of-Hierarchy (Generalization)
- ◆ Attributes
 - ◆ Detection of attributes
 - ◆ Application specific
 - ◆ Attributes in one system can be classes in another system
 - ◆ Turning attributes to classes
- ◆ Methods
 - ◆ Detection of methods
 - ◆ Generic methods: General world knowledge, design patterns
 - ◆ Domain Methods: Dynamic model, Functional model

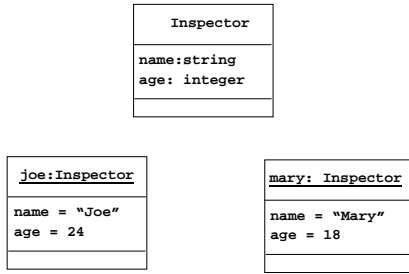
Object vs Class

- ◆ Object (instance): Exactly one thing
 - ◆ The lecture on September 7 on Software Engineering from 9:00 - 10:20
- ◆ A class describes a group of objects with similar properties
 - ◆ IETM, Author, Corrosion, Work order
- ◆ Object diagram: A graphic notation for modeling objects, classes and their relationships ("associations"):
 - ◆ Class diagram: Template for describing many instances of data. Useful for taxonomies, patterns, schemata...
 - ◆ Instance diagram: A particular set of objects relating to each other. Useful for discussing scenarios, test cases and examples
- ◆ Together-J: CASE Tool for building object diagrams, in particular class diagrams
 - ◆ Lecture on September 23

UML: Class and Instance Diagrams

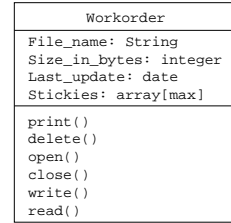


Attributes and Values



Operation, Signature or Method? What when?

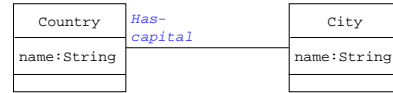
- **Operation:** A function or transformation applied to objects in a class. All objects in a class share the same operations (*Analysis Phase*)
 - **Signature:** Number & types of arguments, type of result value. All methods of a class have the same signature (*Object Design Phase*)
 - **Method:** Implementation of an operation for a class (*Implementation Phase*)
- Polymorphic operation:* The same operation applies to many different classes.



Links and Associations

- Links and associations establish relationships among objects and classes.
- **Link:**
 - A connection between two object instances. A link is like a tuple.
 - A link is an instance of an association
- **Association:**
 - Basically a bidirectional mapping.
 - One-to-one, many-to-one, one-to-many,
 - An association describes a set of links like a class describes a set of objects.

1-to-1 and 1-to-many Associations



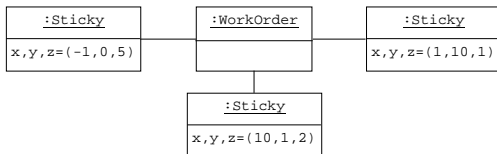
One-to-one association



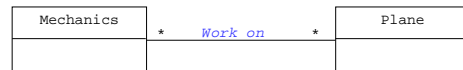
One-to-many association

Object Instance Diagram

Example for 1-to-many

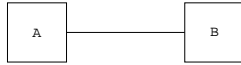


Many-to-Many Associations



Do UML associations have direction?

- ◆ A association between two classes is by default a bi-directional mapping.



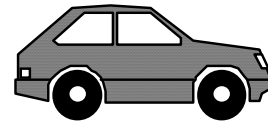
- ◆ Class A can access class B and class B can access class A
- ◆ Both classes play the agent role.



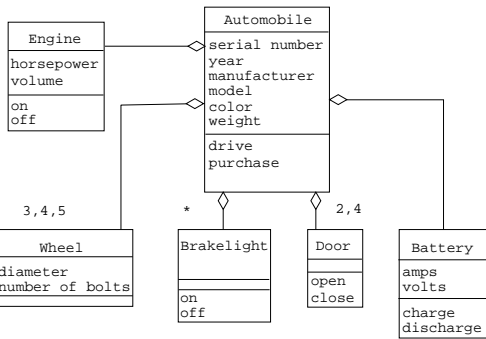
Class A (the "client") accesses class B ("the server"). B is also called *navigable*

Aggregation

- ◆ Models "part of" hierarchy
- ◆ Useful for modeling the breakdown of a product into its component parts (sometimes called bills of materials (BOM) by manufacturers)
- ◆ UML notation: Like an association but with a small diamond indicating the assembly end of the relationship.

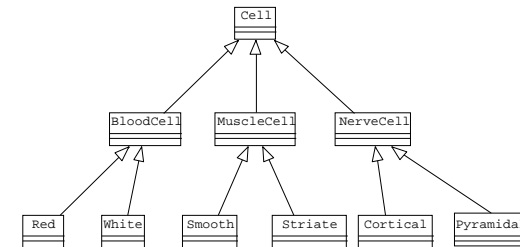


Aggregation



Inheritance

- ◆ Models "kind of" hierarchy
- ◆ Powerful notation for sharing similarities among classes while preserving their differences
- ◆ UML Notation: An arrow with a triangle



Aggregation vs Inheritance

- ◆ Both associations describe trees (hierarchies)
 - ◆ Aggregation tree describes a-part-of relationships (also called and-relationship)
 - ◆ Inheritance tree describes "kind-of" relationships (also called or-relationship)
- ◆ Aggregation relates instances (involves two or more different objects)
- ◆ Inheritance relates classes (a way to structure the description of a single object)

Other Associations

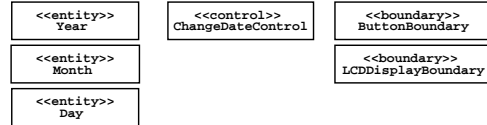
- ◆ Uses:
 - ◆ A subsystem uses another subsystem (System Design)
- ◆ Contains:
 - ◆ Sometimes called "spatial aggregation"
 - ◆ ... contains ...
 - ◆ Example: A UML package contains another UML package
- ◆ Parent/child relationship:
 - ◆ ... is father of ...
 - ◆ ... is mother of ...
- ◆ Seniority:
 - ◆ ... is older than ...
 - ◆ ... is more experienced than ...

Object Types

- Entity Objects
 - Represent the persistent information tracked by the system (Application domain objects, "Business objects")
- Boundary Objects
 - Represent the interaction between the user and the system
- Control Objects:
 - Represent the control tasks performed by the system
- Having three types of objects leads to models that are more resilient to change.
 - The boundary of a system changes more likely than the control
 - The control of the system change more likely than the application domain
- Object types originated in Smalltalk:
 - Model, View, Controller (MV)

Example: 2BWatch Objects

- UML provides several mechanisms to extend the language
- UML provides the stereotype mechanism to present new modeling elements

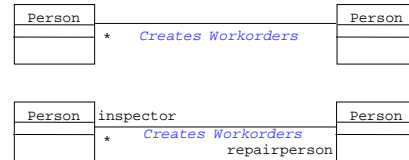


Roles

- A role name is the name that uniquely identifies one end of an association.
- A role name is written next to the association line near the class that plays the role.
- When do you use role names?
 - Necessary for associations between two objects of the same class
 - Also useful to distinguish between two associations between the same pair of classes
- When do you not use role names?
 - If there is only a single association between a pair of distinct classes, the names of the classes serve as good role names

Example of Role

Problem Statement : A person assumes the role of repairer with respect to another person, who assumes the role of inspector with respect to the first person.



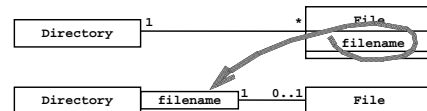
Roles in Associations

- Client Role:
 - An object that can operate upon other objects but that is never operated upon by other objects.
- Server Role:
 - An object that never operates upon other objects. It is only operated upon by other objects.
- Agent Role:
 - An object that can both operate upon other objects and be operated upon by other objects. An agent is usually created to do some work on behalf of an actor or another agent.

Qualification

- The qualifier improves the information about the multiplicity of the association between the classes.
- It is used for reducing 1-to-many multiplicity to 1-1 multiplicity

Without qualification: A directory has many files. A file belongs only to one directory.



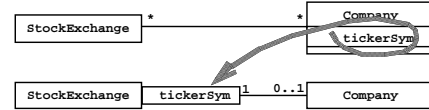
With qualification: A directory has many files, each with a unique name

Example

Problem Statement : A stock exchange lists many companies. However, a stock exchange lists only one company with a given ticker symbol. A company may be listed on many stock exchanges, possibly with different ticker symbols. Find company with ticker symbol AAPL.



Use of Qualification reduces multiplicity



How do you find classes?

- ◆ Learn about problem domain: Observe your client
- ◆ Apply general world knowledge and intuition
- ◆ Take the flow of events and find participating objects in use cases
- ◆ Apply design patterns
- ◆ Try to establish a taxonomy
- ◆ Do a textual analysis of scenario or flow of events (Abbott Textual Analysis, 1983)
- ◆ Nouns are good candidates for classes

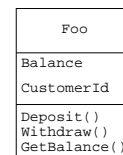
Mapping parts of speech to object model components [Abbot 1983]

Part of speech	Model component	Example
Proper noun	object	Jim Smith
Improper noun	class	Toy, doll
Doing verb	method	Buy, recommend
being verb	inheritance	is-a (kind-of)
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	enter
intransitive verb	method (event)	depends on

Example: Scenario from Problem Statement

- ◆ Jim Smith enters a store with the intention of buying a toy for his 3 year old child.
- ◆ Help must be available within less than one minute.
- ◆ The store owner gives advice to the customer. The advice depends on the age range of the child and the attributes of the toy.
- ◆ Jim selects a dangerous toy which is unsuitable for the child.
- ◆ The store owner recommends a more yellow doll.

Object Modeling in Practice: Class Identification



Class Identification: Name of Class, Attributes and Methods

Object Modeling in Practice: Encourage Brainstorming

Naming is important!

Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 37

Object Modeling in Practice

Find New Objects
Iterate on Names, Attributes and Methods

Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 38

Object Modeling in Practice: A Banking System

Find New Objects
Iterate on Names, Attributes and Methods
Find Associations between Objects
Label the associations
Determine the multiplicity of the associations

Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 39

Object Modeling in Practice: Categorize!

Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 40

Avoid Ravioli Models

**Don't put too many classes into the same package:
7+-2 (or even 5+-2)**

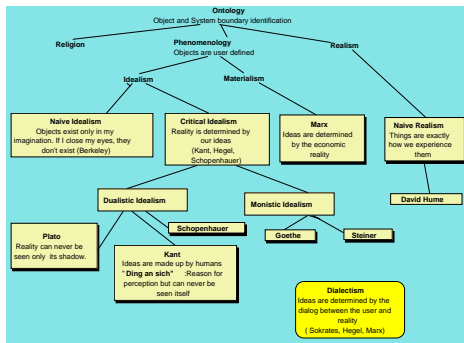
Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 41

Object Modeling in Practice: Heuristics

- ◆ Explicitly schedule meetings for object identification
- ◆ Try to differentiate between entity, boundary and control objects
- ◆ Find associations and their multiplicity
 - ◆ Unusual multiplicities usually lead to new objects or categories
- ◆ Identify Aggregation
- ◆ Identify Inheritance: Look for a Taxonomy, Categorize
- ◆ Allow time for brainstorming , Iterate, iterate

Bertal Bruggen & Allen Dettl | Object-Oriented Software Engineering: Conquering Complex and Changing Systems | 42

Software Engineers are not the only System Analysts



Bernd Bruggen & Albin Dettl

Object-Oriented Software Engineering: Conquering Complex and Changing Systems

43

What is a Software Engineer?

♦ From the point of view of phenomenology, Software Engineers are dialectic monistic idealists:

- ♦ **Idealists:**
 - ♦ They accept that ideas (called requirements or “customer’s wishlist”) are different from reality.
- ♦ **They are not realists:**
 - ♦ The reality might not yet exist (“Vaporware is always possible”)
- ♦ **They are monistic:**
 - ♦ They are optimistic that their ideas can describe reality (“das Ding an sich”).
- ♦ **Dialectic:**
 - ♦ They do this in a dialogue with the customer

Bernd Bruggen & Albin Dettl

Object-Oriented Software Engineering: Conquering Complex and Changing Systems

44

Summary

In this lecture, we reviewed the construction of the object model from use case model. In particular, we described:

- ♦ Identification of objects
- ♦ Refinement of objects with attributes and operations
- ♦ Generalization of concrete classes
- ♦ Identification of associations
- ♦ Reduction of multiplicity using qualification.

In the next lecture, we describe the construction of the dynamic model from the use case and object models.

Bernd Bruggen & Albin Dettl

Object-Oriented Software Engineering: Conquering Complex and Changing Systems

45