System Modeling

Introduction into Software Engineering
Lecture 5
2 May 2007

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

Outline of the next lectures

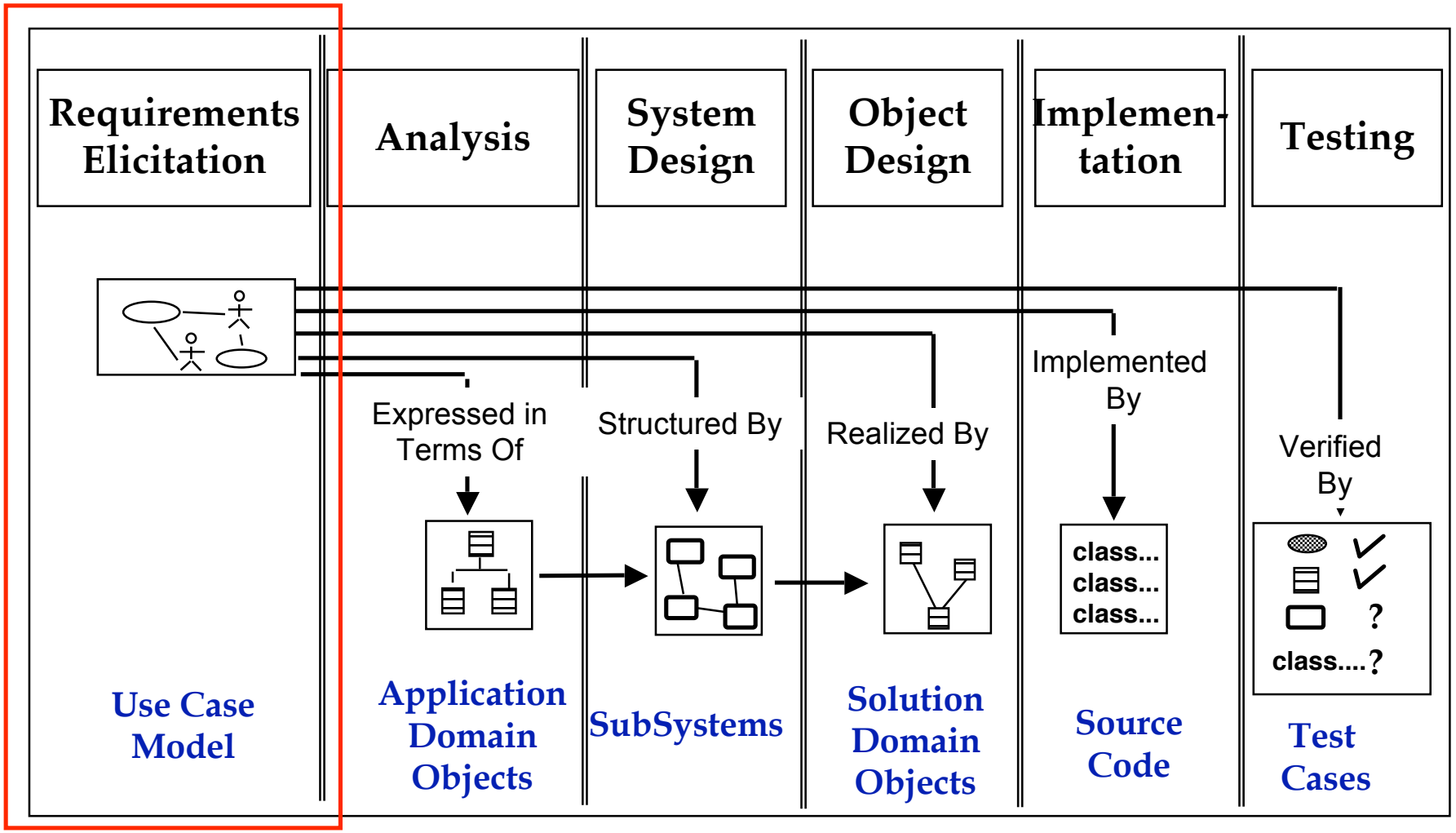
System Modeling =

- Functional Modeling
- Object Modeling
- Dynamic Modeling

Outline of this lecture

Functional Modeling
- Scenarios
- Use Cases
  - Finding Use Cases
  - Flow of Events
  - Use Case Associations
  - Use Case Refinement
- Object Modeling:
  - From use cases to class diagrams
  - Activities during object modeling
  - Object identification
  - Object types: Entity, boundary and control objects
  - Object naming
  - Abott's technique helps in object identification
  - Users of class diagrams
- Summary

Software lifecycle activities



Scenarios

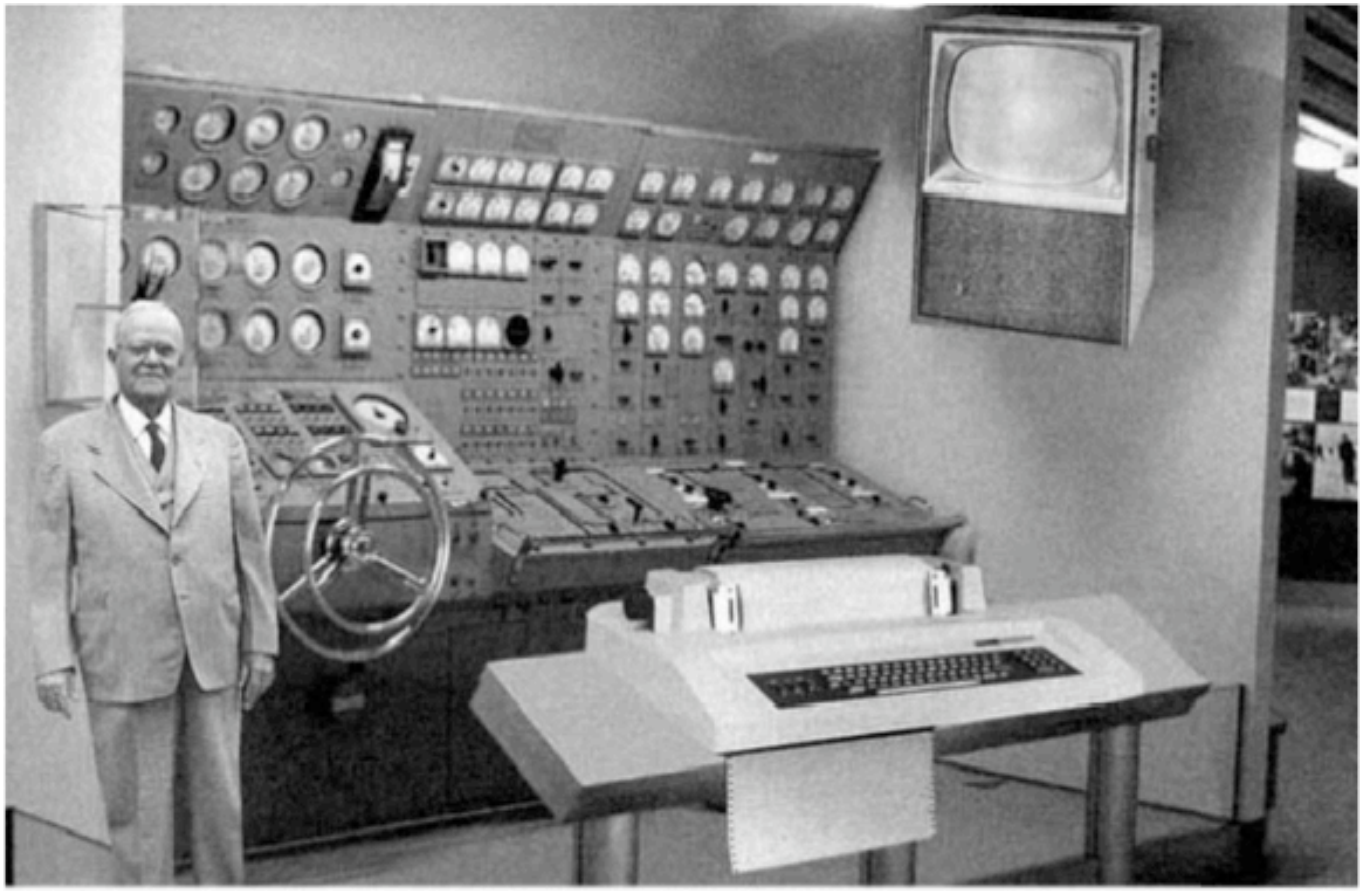
- Scenario: "A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carroll, Scenario-Based Design, Wiley, 1995]
- A concrete, focused, informal description of a single feature of the system used by a single actor.
- Scenarios can have many different uses during the software lifecycle
  - ***Requirements Elicitation***: As-is scenario, visionary scenario
  - ***Client Acceptance Test:*** Evaluation scenario
  - ***System Deployment:*** Training scenario

Types of Scenarios

- As-is scenario:
  - Describes a current situation. Usually used in re-engineering projects. The user describes the system
    - Example: Description of Letter-Chess
- Visionary scenario:
  - Describes a future system. Usually used in greenfield engineering and reengineering projects
  - Can often not be done by the user or developer alone
    - Example: Description of an interactive internet-based Tic Tac Toe game tournament
    - Example: Description - in the year 1954 - of the Home Computer of the Future.

Example of A Visionary Scenario

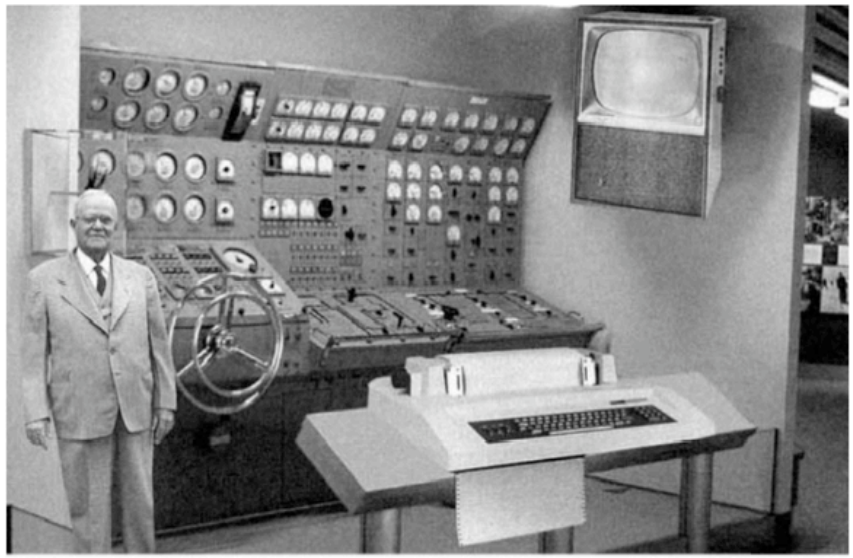
- Published in the Popular Mechanic, 1954
- Vision of the The Home Computer in 2004



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

It is a Hoax!

- Source of original picture:
  - A Smithsonian Institution exhibit *Fast Attacks and Boomers: Submarines in the Cold War* on April 11, 2000.
  - http://www.chinfo.navy.mil/navpalib/cno/n87/usw/issue_8/smithsonian.html
- The orginal shows the maneuvering room of the USS Batfish SSN-681.
  - The control display was replaced with an old plotter, the TV was "aged" with Photoshop,
  - The RAND scientist is probably really from 1954.



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Lessons learned

1. Trust your sources
   - Talk to the Application domain expert, not to You Tube
2. Use can use scetches and tools to create a visionary scenario
   - Photos
     - Touchup with tools such as Photoshop
   - Movies
     - Software Cinema: A technique to illustrate scenarios with a a film (Oliver Creighton, 2005)
3. Be honest when you create a visionary scenario
   - Name your sources.

Types of Scenarios (2)

- Evaluation scenario:
  - User tasks against which the system is to be evaluated.
    - Example: Four users (two novice, two experts) play in a TicTac Toe tournament in ARENA.
- Training scenario:
  - Step by step instructions that guide a novice user through a system
    - Example: How to play Tic Tac Toe in the ARENA Game Framework.

How do we find scenarios?

- Don’t expect the client to be verbal if the system does not exist
  - Client understands problem domain, not the solution domain.
- Don’t wait for information even if the system exists
  - “What is obvious does not need to be said”
- Engage in a dialectic approach
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed

Scenario example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and John

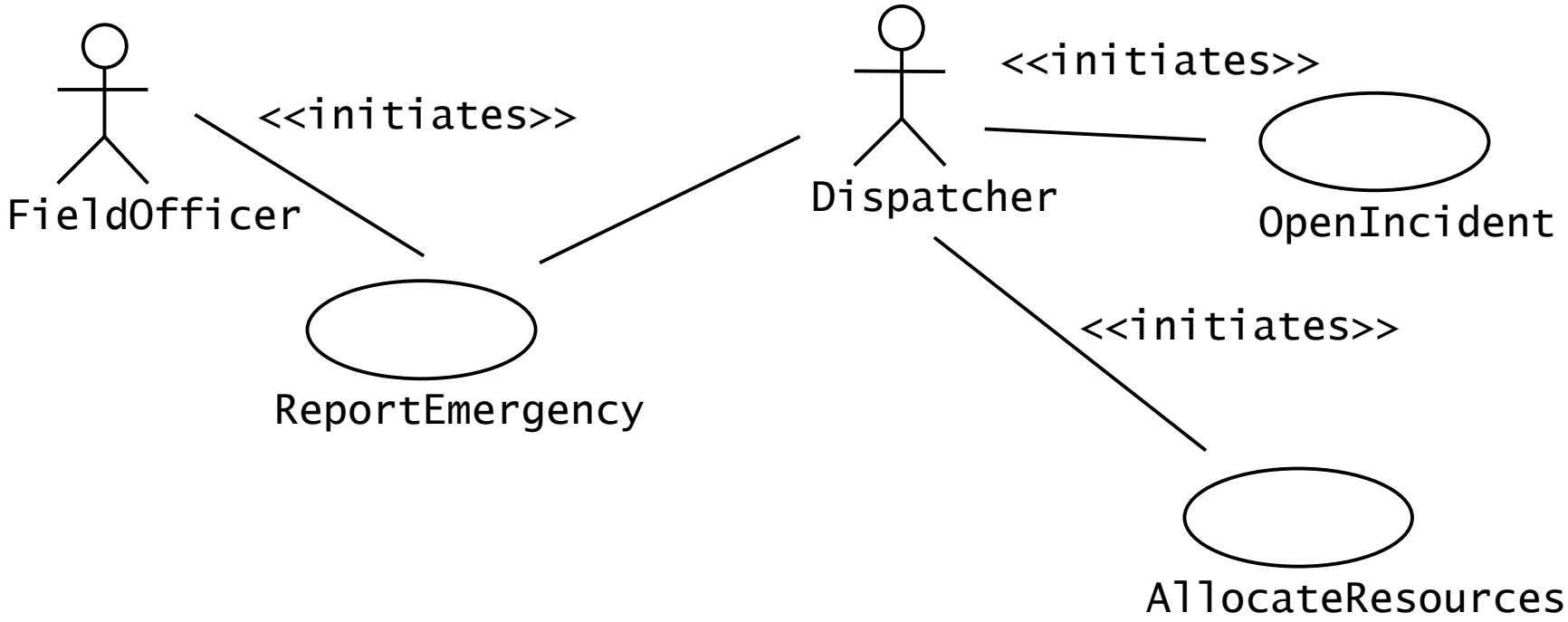
Heuristics for finding scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on questions alone
- Insist on task observation if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user, not just to the client
  - Expect resistance and try to overcome it.

After the scenarios are formulated

- Find all the use cases in the scenario that specify all instances of how to report a fire
  - Example: "Report Emergency" in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the entry condition
  - Describe the flow of events
  - Describe the exit condition
  - Describe exceptions
  - Describe nonfunctional requirements

Use Case Model for Incident Management



How to find Use Cases

- Select a narrow vertical slice of the system (i.e. one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)

Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The FieldOfficer is notified immediately if the connection between terminal and central is lost.
  - The Dispatcher is notified immediately if the connection between a FieldOfficer and central is lost.
- Flow of Events: **on next slide.**
- Special Requirements:
  - The FieldOfficer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

Use Case Example: ReportEmergency Flow of Events

1. The **FieldOfficer** activates the "Report Emergency" function of her terminal. FRIEND responds by presenting a form to the officer.
2. The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes a response to the emergency situation. Once the form is completed, the FieldOfficer submits the form, and the **Dispatcher** is notified.
3. The Dispatcher creates an Incident in the database by invoking the OpenIncident use case. He selects a response and acknowledges the report.
4. The FieldOfficer receives the acknowledgment and the selected response.

Order of steps when formulating use cases

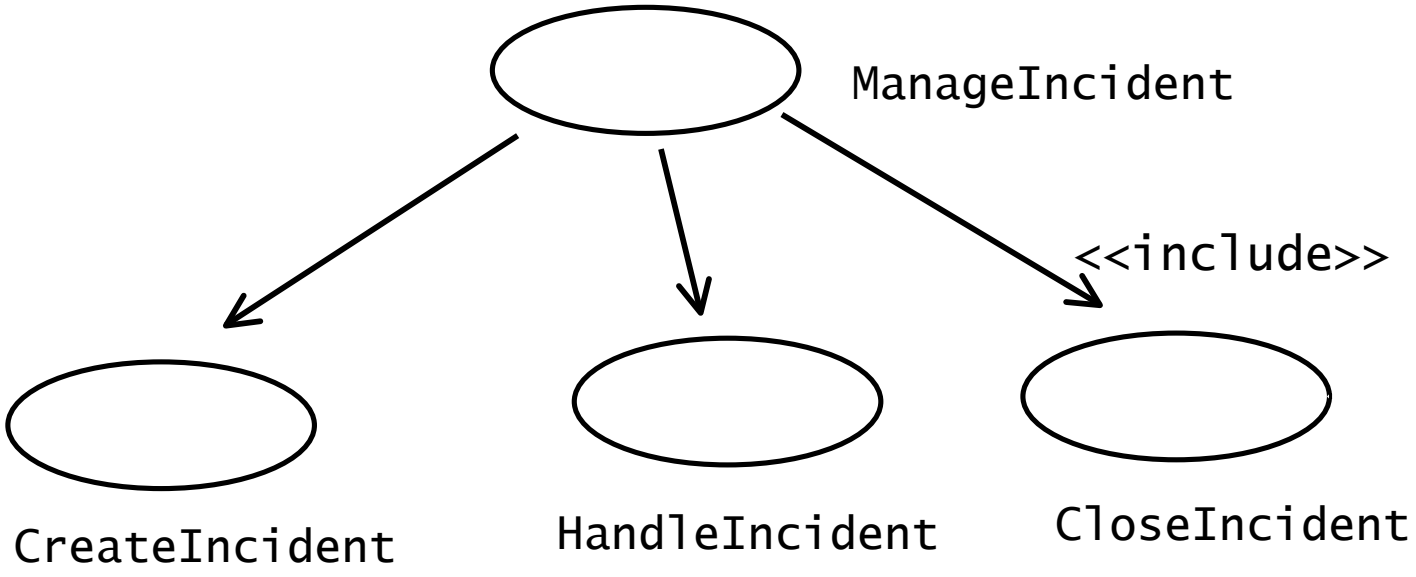
- First step: Name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names ("Bob") to participating actors ("Field officer")
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: Concentrate on the flow of events
  - Use informal natural language

Use Case Associations

- Dependencies between use cases are represented with use case associations
- Associations are used to reduce complexity
  - Decompose a long use case into shorter ones
  - Separate alternate flows of events
  - Refine abstract use cases
- Types of use case associations
  - Includes
  - Extends
  - Generalization

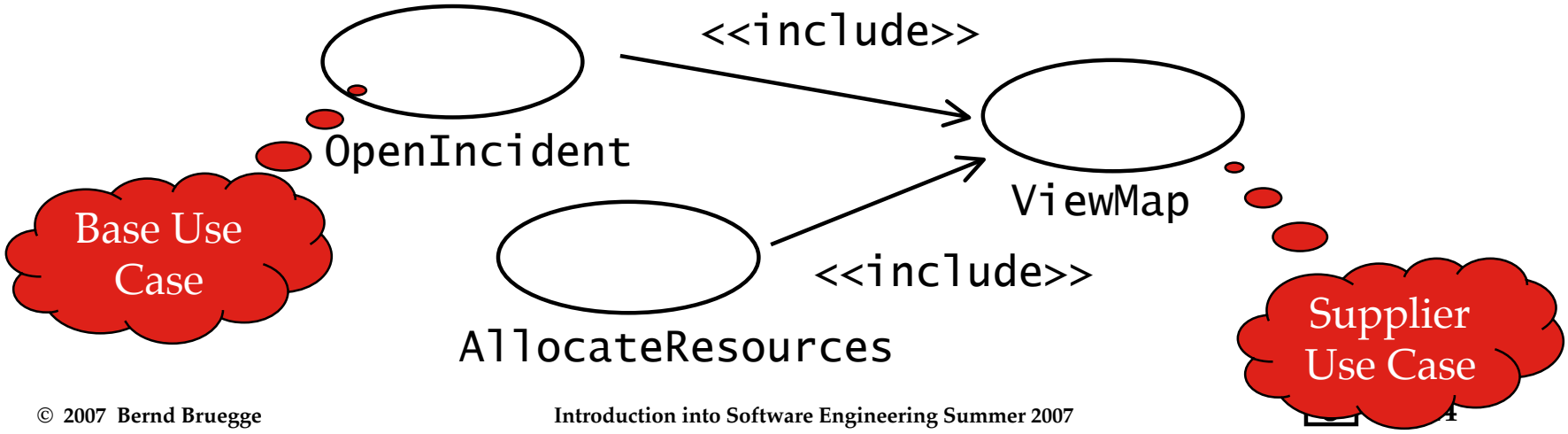
<<include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into shorter use cases



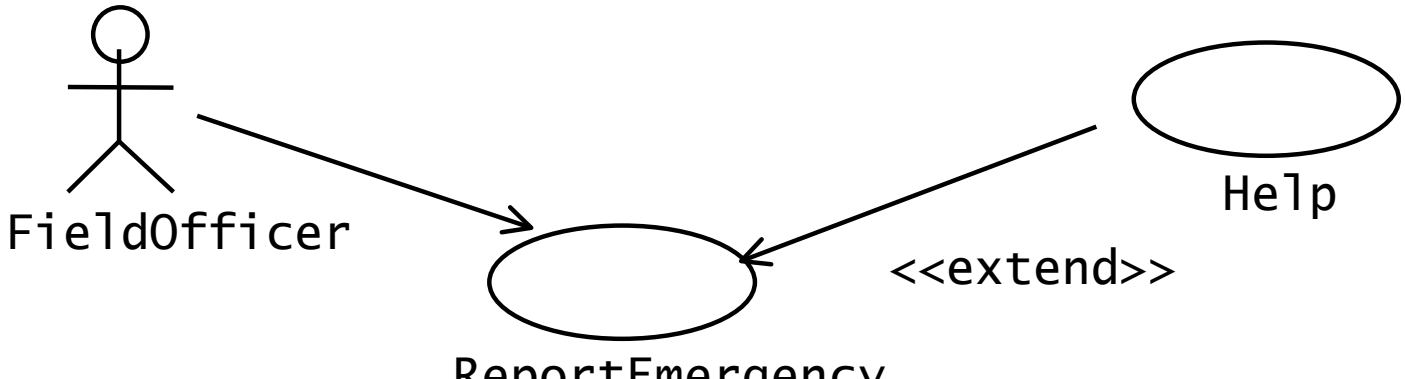
<<include>>: Reuse of Existing Functionality

- Problem: There are overlaps among use cases. How can we *reuse* flows of events instead of duplicating them?
- Solution: The *includes association* from use case A to use case B indicates that an instance of use case A performs all the behavior described in use case B ("A delegates to B")
- Example: Use case "ViewMap" describes behavior that can be used by use case "OpenIncident" ("ViewMap" is factored out)



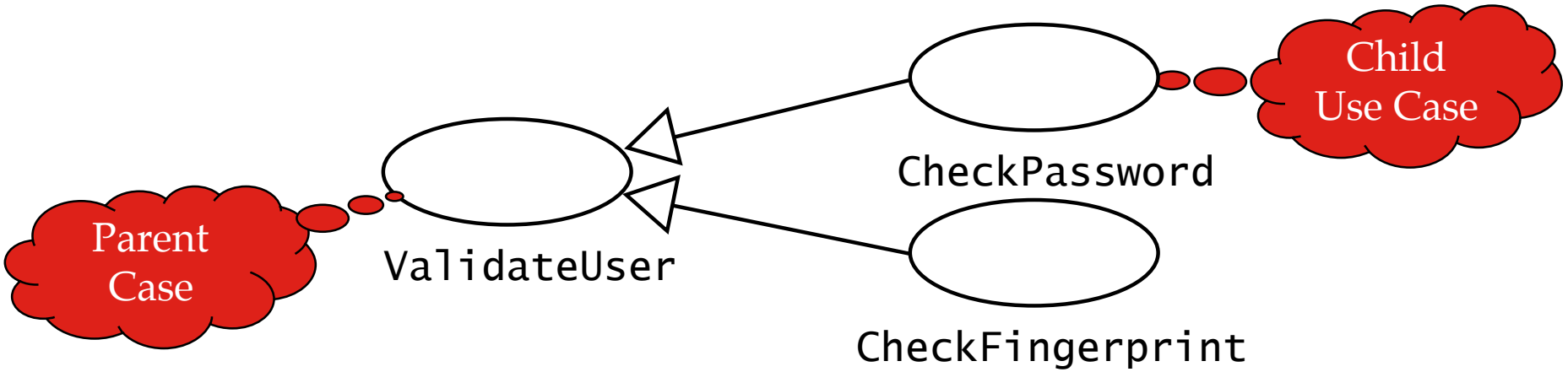
<<extend>> Association for Use Cases

- Problem: The functionality in the original problem statement needs to be extended.
- Solution: An *extend association* from use case A to use case B
- Example: "ReportEmergency" is complete by itself, but can be extended by use case "Help" for a scenario in which the user requires help



Generalization in Use Cases

- Problem: We want to factor out common (but not identical) behavior.
- Solution: The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- Example: "ValidateUser" is responsible for verifying the identity of the user. The customer might require two realizations: "CheckPassword" and "CheckFingerprint"



Another Use Case Example

Actor **Bank Customer**

- Person who owns one or more Accounts in the Bank.

Withdraw Money

- The Bank Customer specifies a Account and provides credentials to the Bank proving that s/he is authorized to access the Bank Account.
- The Bank Customer specifies the amount of money s/he wishes to withdraw.
- The Bank checks if the amount is consistent with the rules of the Bank and the state of the Bank Customer's account. If that is the case, the Bank Customer receives the money in cash.

Use Case Attributes

Use Case **Withdraw Money Using ATM**

Initiatiating actor:

- Bank Customer

Preconditions:

- Bank Customer has opened a Bank Account with the Bank ***and***
- Bank Customer has received an ATM Card and PIN

Postconditions:

- Bank Customer has the requested cash ***or***
- Bank Customer receives an explanation from the ATM about why the cash could not be dispensed

Use Case Flow of Events

Actor steps	System steps
1.The Bank Customer inputs the card into the ATM.	
	2.The ATM requests the input of a four-digit PIN.
3. The Bank Customer types in PIN.	
	4. If several accounts are recorded on the card, the ATM offers a choice of the account numbers for selection by the Bank Customer
5. The Bank Customer selects an account.	
	6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn.
7. The Bank Customer inputs an amount.	
	8.The ATM outputs the money and a receipt and stops the interaction.

Use Case Exceptions

Actor steps

1. The Bank Customer inputs her card into the ATM.**[Invalid card]**

3. The Bank Customer types in PIN. **[Invalid PIN]**

5. The Bank Customer selects an account .

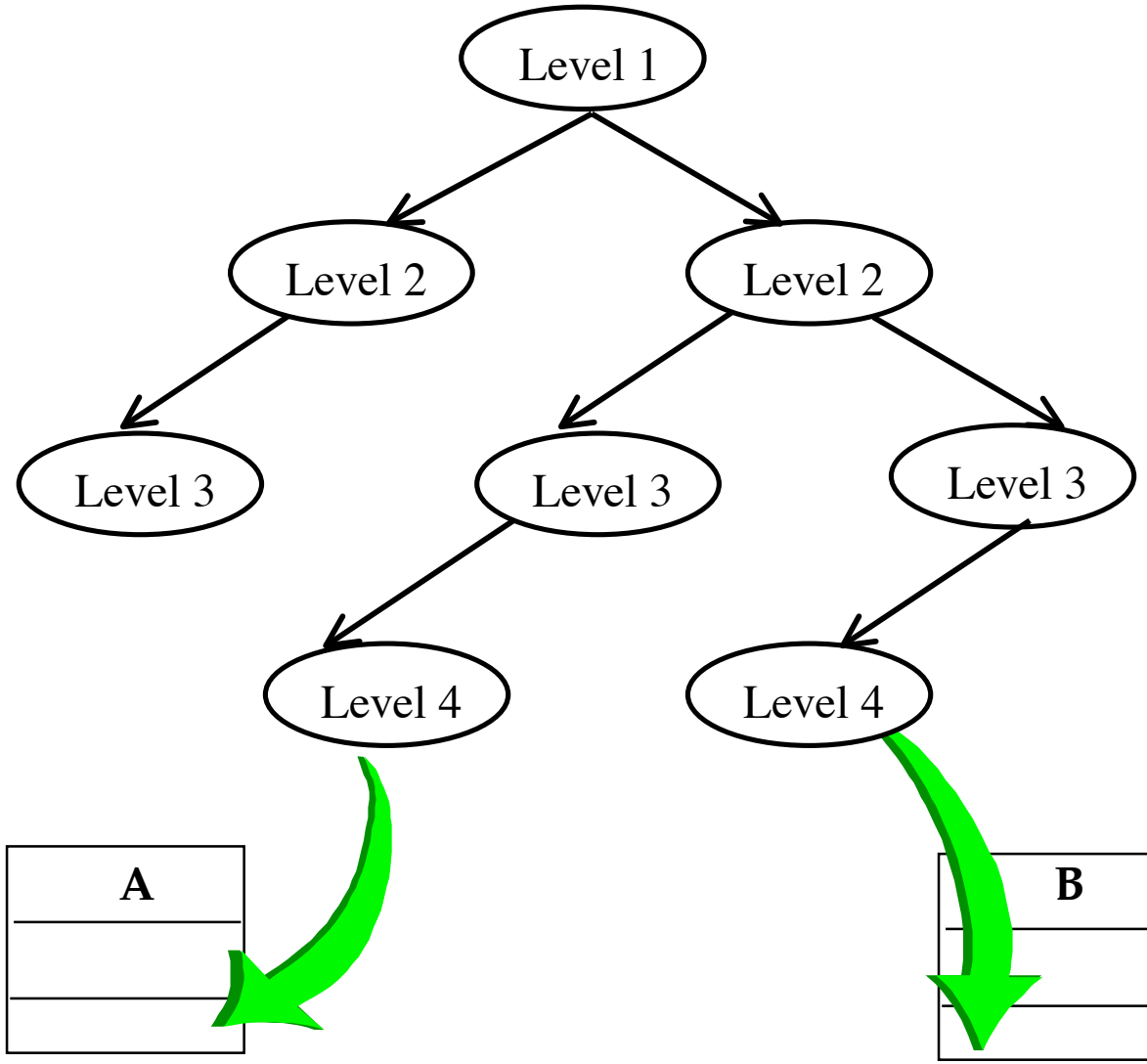
7. The Bank Customer inputs an amount. **[Amount over limit]**

[Invalid card]
The ATM outputs the card and stops the interaction.

[Invalid PIN]
The ATM announces the failure and offers a 2nd try as well as canceling the whole use case. After 3 failures, it announces the possible retention of the card. After the 4th failure it keeps the card and stops the interaction.

[Amount over limit]
The ATM announces the failure and the available limit and offers a second try as well as canceling the whole use case.

From Use Cases to Objects



Top Level Use Case

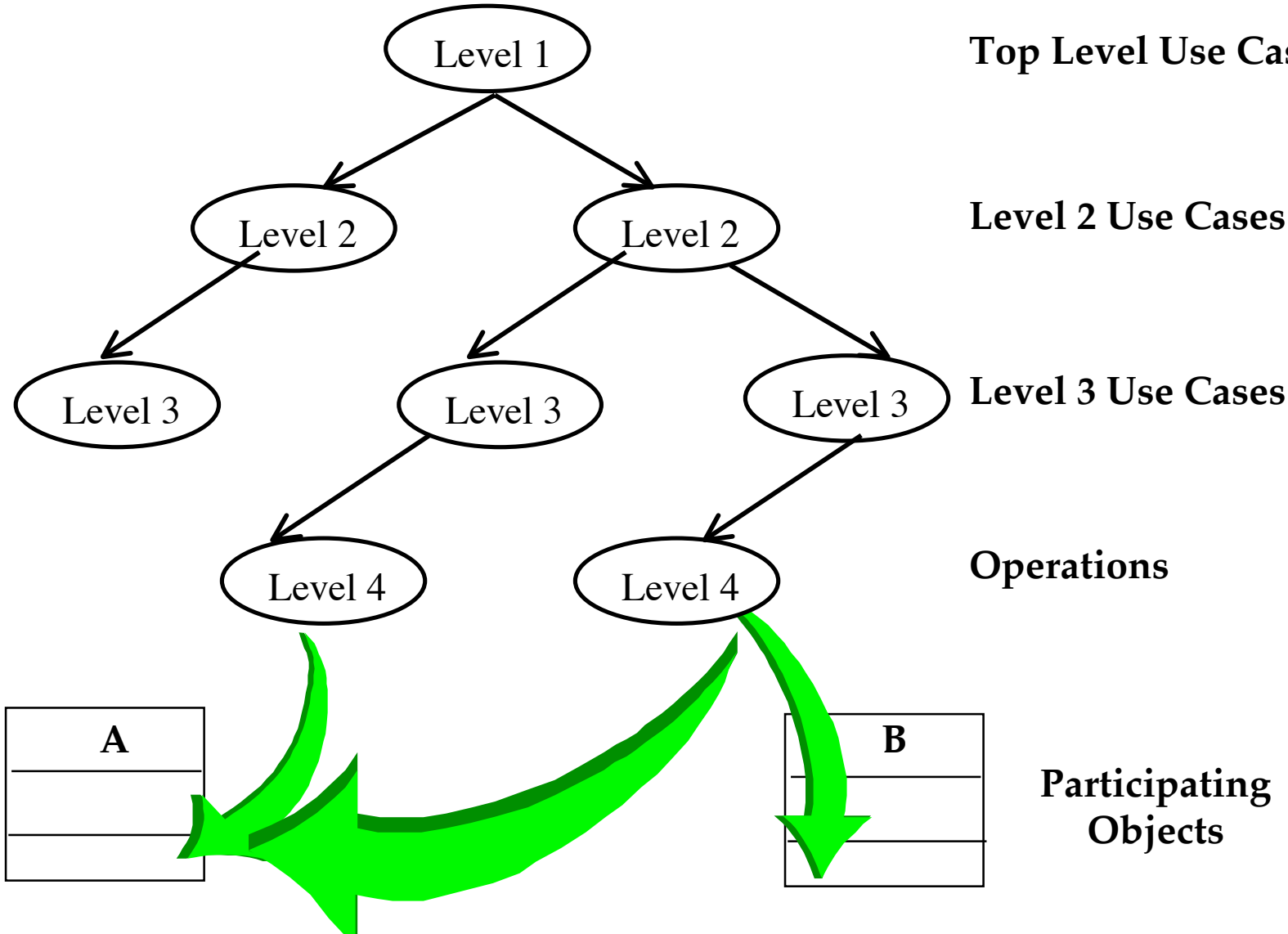
Level 2 Use Cases

Level 3 Use Cases

Operations

A and B are called Participating Objects

Use Cases used by more than one Object



Guidelines for Formulation of Use Cases (1)

- Name
  - Use a verb phrase to name the use case.
  - The name should indicate what the user is trying to accomplish.
  - Examples:
    - "Request Meeting", "Schedule Meeting", "Propose Alternate Date"
- Length
  - A use case description should not exceed 1-2 pages. If longer, use include relationships.
  - A use case should describe a complete set of interactions.

Guidelines for Formulation of Use Cases (2)

Flow of events:

- Use the active voice. Steps should start either with "The Actor" or "The System …".
- The causal relationship between the steps should be clear.
- All flow of events should be described (not only the main flow of event).
- The boundaries of the system should be clear. Components external to the system should be described as such.
- Define important terms in the glossary.

Example of a badly written Use Case

"The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through."

- What is wrong with this use case?

Example of a badly written Use Case

"The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through."

- It contains no actors
- It is not clear which action triggers the ticket being issued
- Because of the passive form, it is not clear who opens the gate
  - The driver? The computer? A gate keeper?
- It is not a complete transaction.
  - A complete transaction would also describe the driver paying for the parking and driving out of the parking lot.

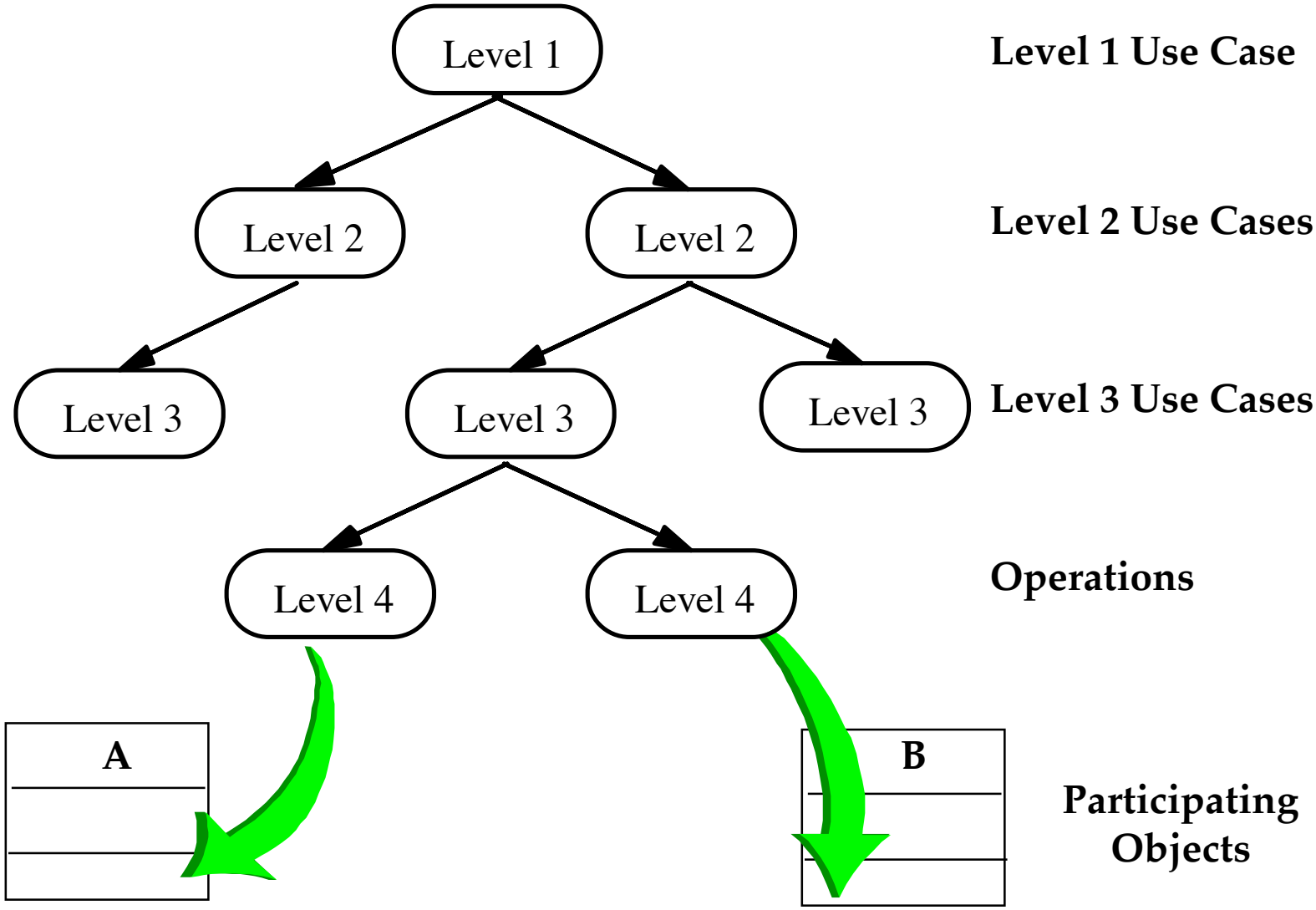
How to write a use case (Summary)

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - "This use case starts when…"
- Flow of Events
  - Free form, informal natural language
- Exit condition
  - "This use cases terminates when…"
- Exceptions
  - Describe what happens if things go wrong
- Special Requirements
  - Nonfunctional Requirements, Constraints

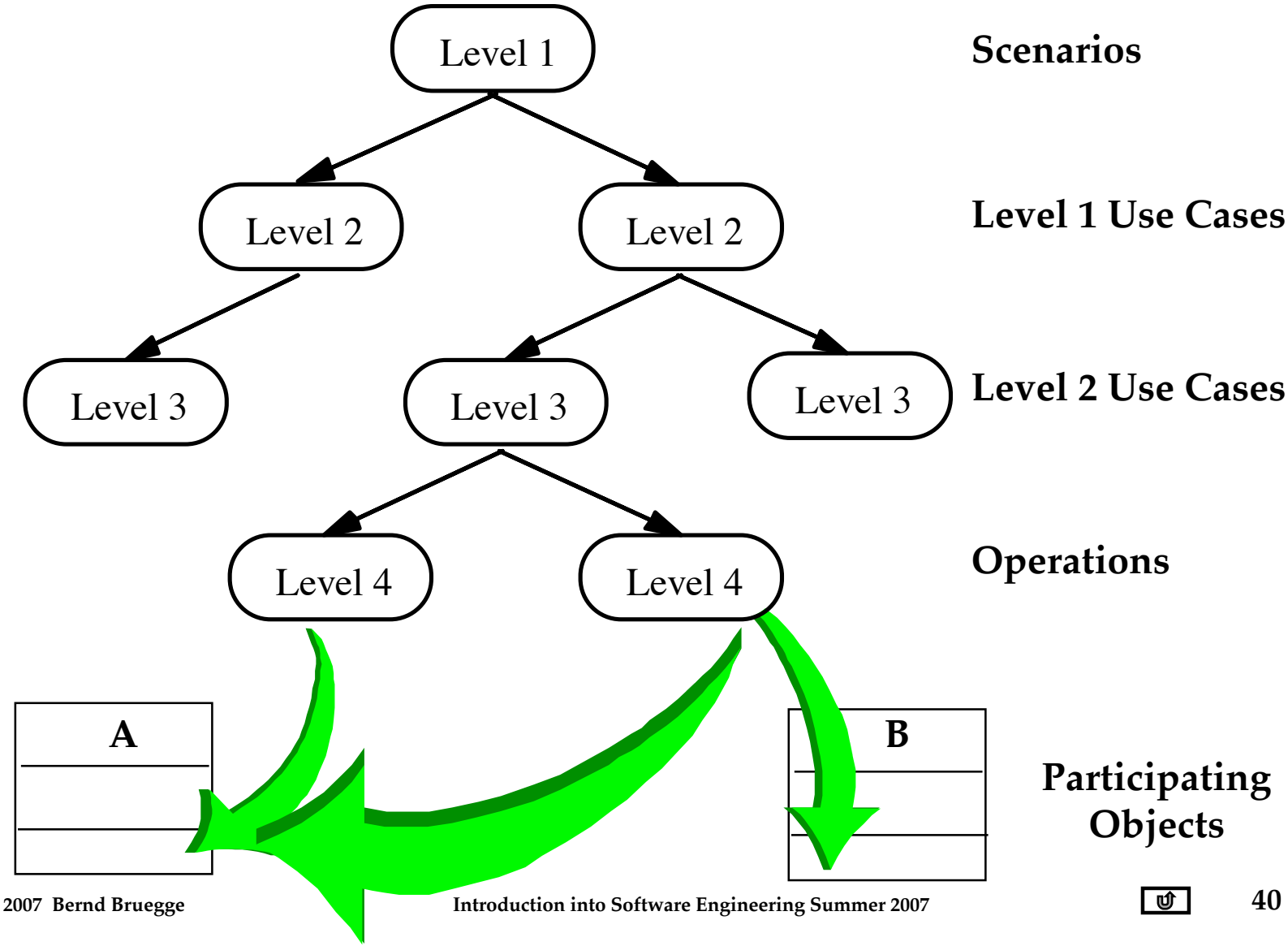
Summary

- Scenarios:
  - Great way to establish communication with client
  - Different types of scenarios: As-Is, visionary, evaluation and training
- Use cases
  - Abstractions of scenarios
- Use cases bridge the transition between functional requirements and objects.

From Use Cases to Objects



From Use Cases to Objects: Why Functional Decomposition is not Enough



Activities during Object Modeling

Main goal: Find the important abstractions

- 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
- What happens if we find the wrong abstractions?
  - We iterate and revise the model

Class Identification

Class identification is crucial to object-oriented modeling

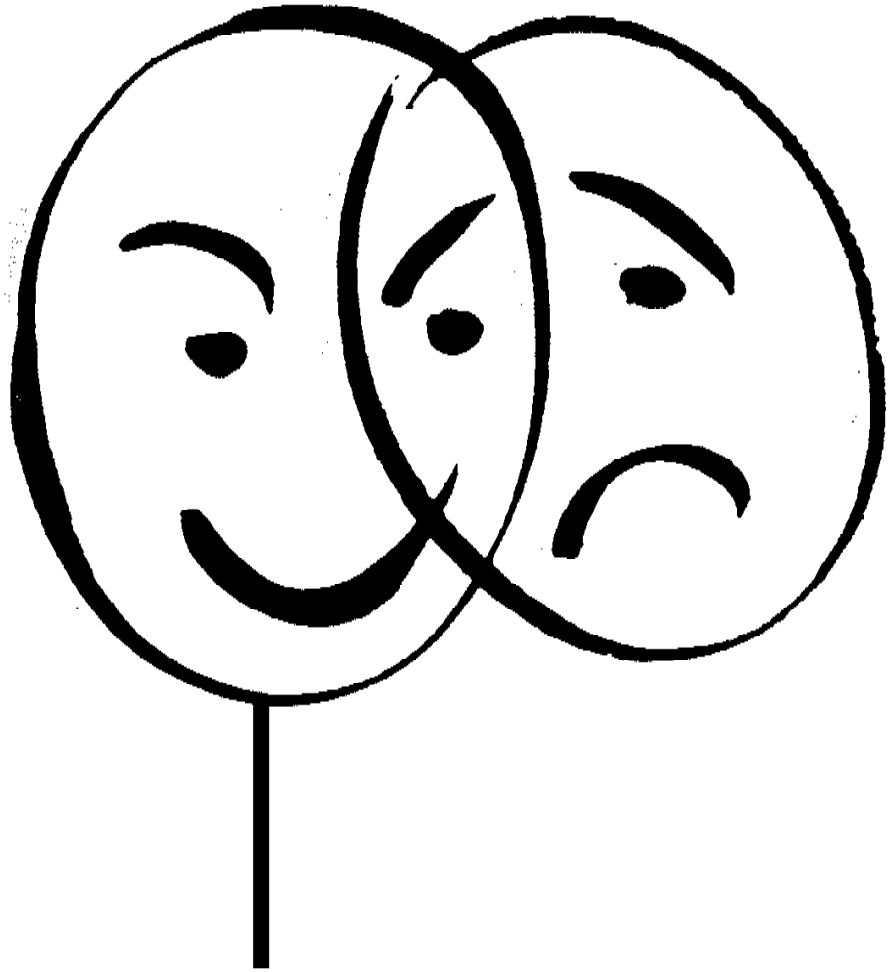
- Helps to identify the important entities of a system

- 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

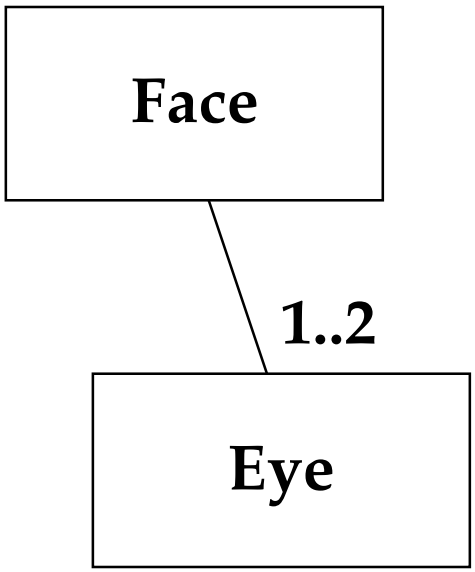
Class identification is an ancient problem

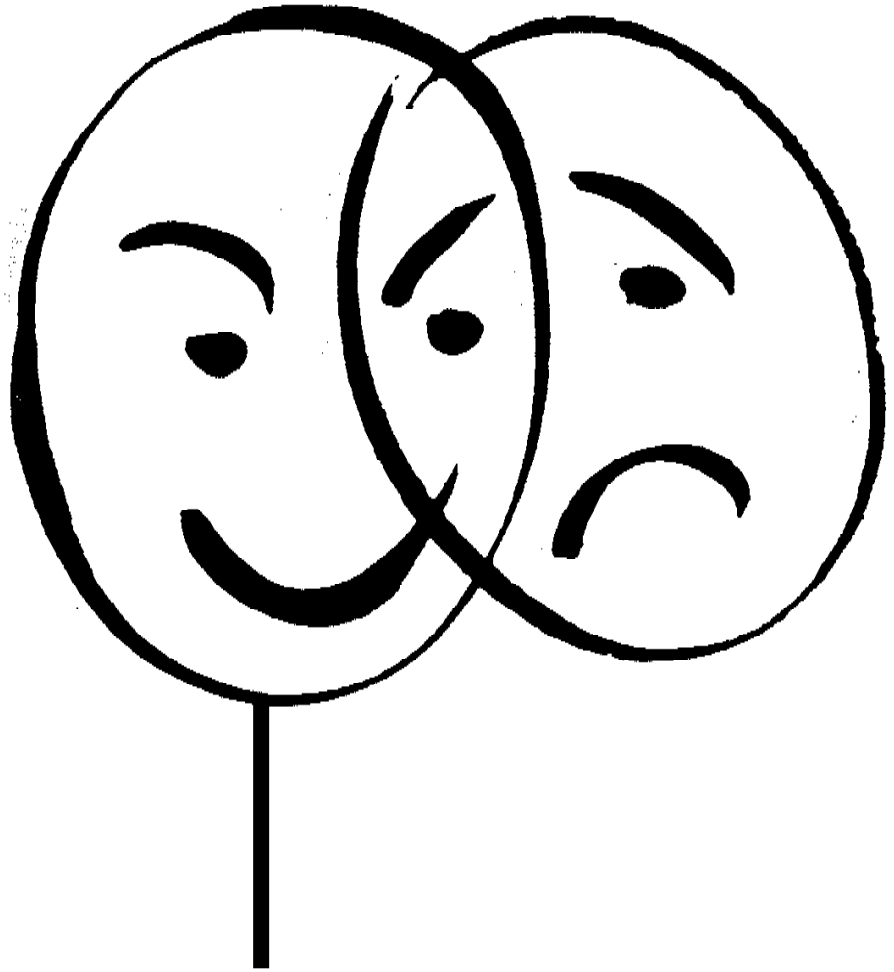
- Objects are not just found by taking a picture of a scene or domain
- The application domain has to be analyzed.
- Depending on the purpose of the system different objects might be found
  - How can we identify the purpose of a system?
  - Scenarios and use cases
- Another important problem: Define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

What is This?



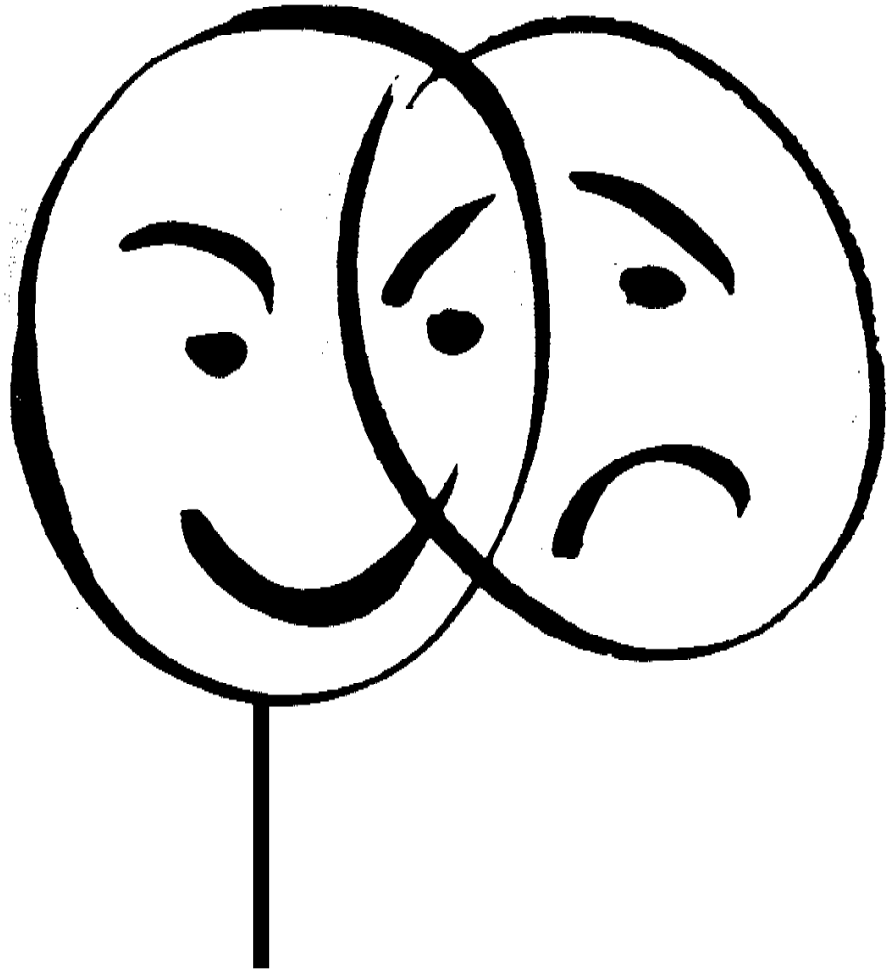
What is This?





Modeling in Action

- Face
- Sad
- Happy
- Is it one Face or two?
- Mask
- Who is using it?
  - Person at Carneval?
  - Bankrobber?
  - Painting collector
- How is it used?



Pieces of an Object Model

- Classes
- Associations (Relations)
- Attributes
- Operations

Associations

- Types of Associations
  - Canonical associations
    - Part-of Hierarchy (Aggregation)
    - Kind-of Hierarchy (Inheritance)
  - Generic associations

Attributes

- Detection of attributes is application specific
- Attributes in one system can be classes in another system
- Turning attributes to classes and vice versa

Operations

- Source of operations
  - Use cases in the functional model
  - General world knowledge
  - Generic operations: Get/Set
  - Design Patterns
  - Application domain specific operations
  - Actions and activities in the dynamic model

Object vs Class

- Object (instance): Exactly one thing
  - This lecture on object modeling
- A class describes a group of objects with similar properties
  - Game, Tournament, mechanic, car, database
- Object diagram: A graphical notation for modeling objects, classes and their relationships
  - Class diagram: Template for describing many instances of data. Useful for taxonomies, patters, schemata...
  - Instance diagram: A particular set of objects relating to each other. Useful for discussing scenarios, test cases and examples

Class Identification

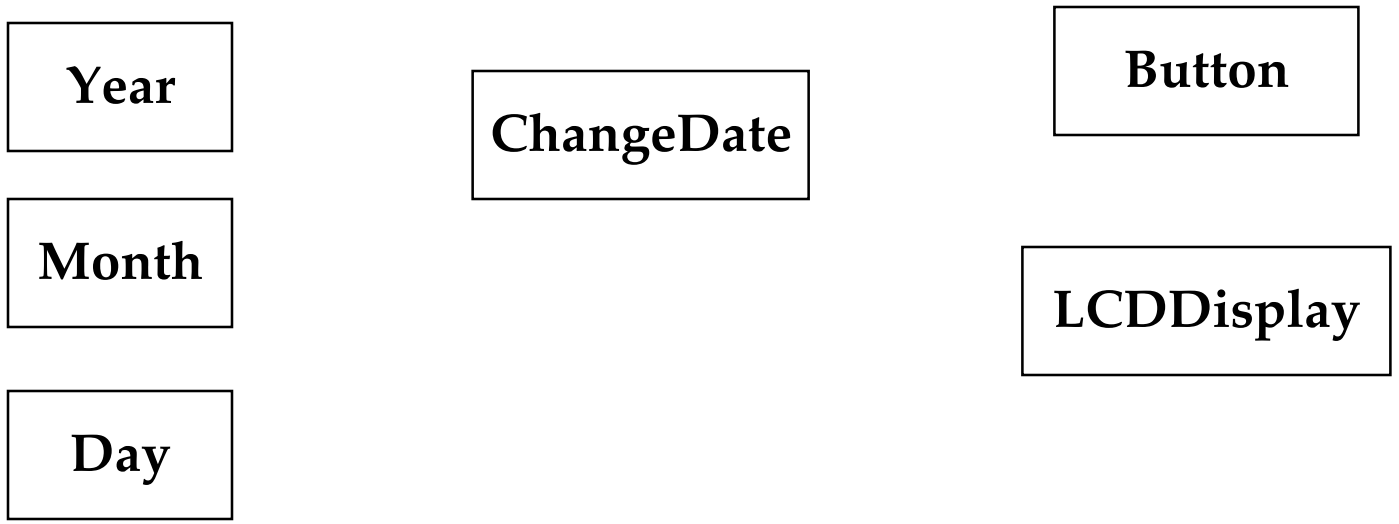
- Approaches
  - Application domain approach
    - Ask application domain experts to identify relevant abstractions
  - Syntactic approach
    - Start with use cases
    - Analyze the text to identify the objects
    - Extract participating objects from flow of events
  - Design patterns approach
    - Use reusable design patterns
  - Component-based approach
    - Identify existing solution classes.

There are different types of Objects

- Entity Objects
  - Represent the persistent information tracked by the system (Application domain objects, also called "Business objects")
- Boundary Objects
  - Represent the interaction between the user and the system
- Control Objects
  - Represent the control tasks performed by the system.

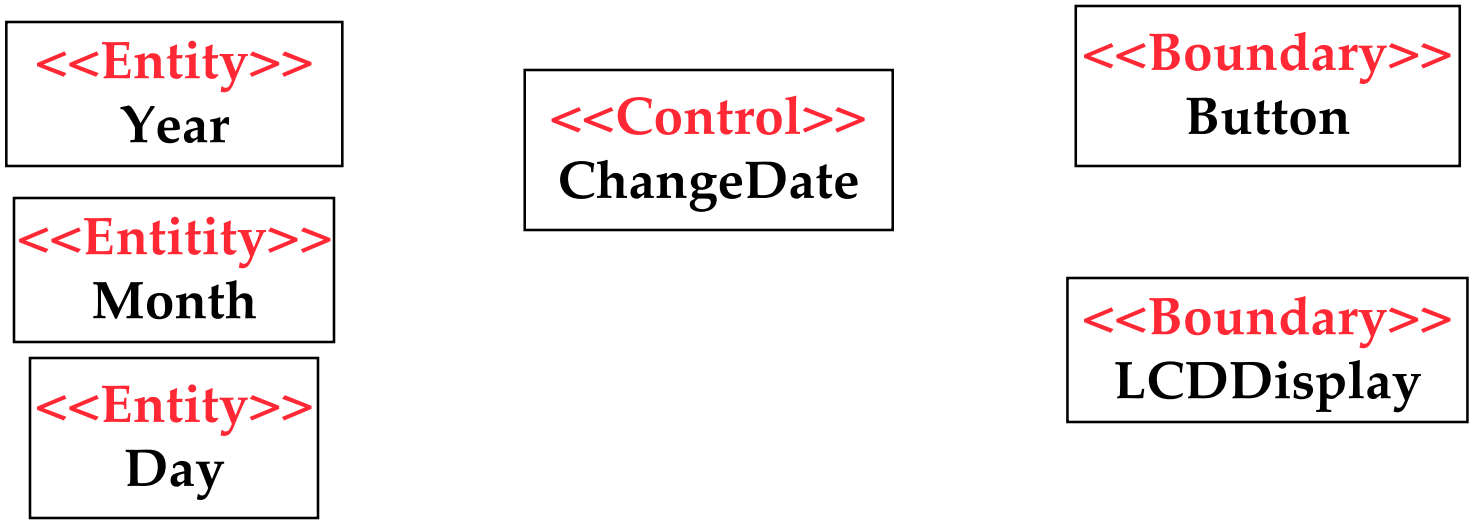
Example: 2BWatch Modeling

To distinguish these different object types in the model we can use the UML Stereotype mechanism



Naming Object Types in UML

- UML provides the stereotype mechanism to introduce new types of modeling elements
- UML is an extensible language



Object Types and Change

- Having three types of object leads to models that are more resilient to change
  - The interface of a system changes more likely than the control
  - The way the system is controlled changes more likely than the application domain
- Object types originated in Smalltalk:
  - Model, View, Controller (MVC)
  - Entity, Boundary, Control Objects
- Next topic: Finding objects.

Finding Participating Objects in Use Cases

- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis)
  - Nouns are candidates for objects/classes
  - Verbs are candidates for operations
  - Also called Abbott's Technique

- After objects/classes are found, identify their types
  - Identify real world entities that the system needs to keep track of (FieldOfficer → entity object)
  - Identify real world procedures that the system needs to keep track of (EmergencyPlan → control object)
  - Identify interface artifacts (PoliceStation → boundary object)

Example for using the Technique

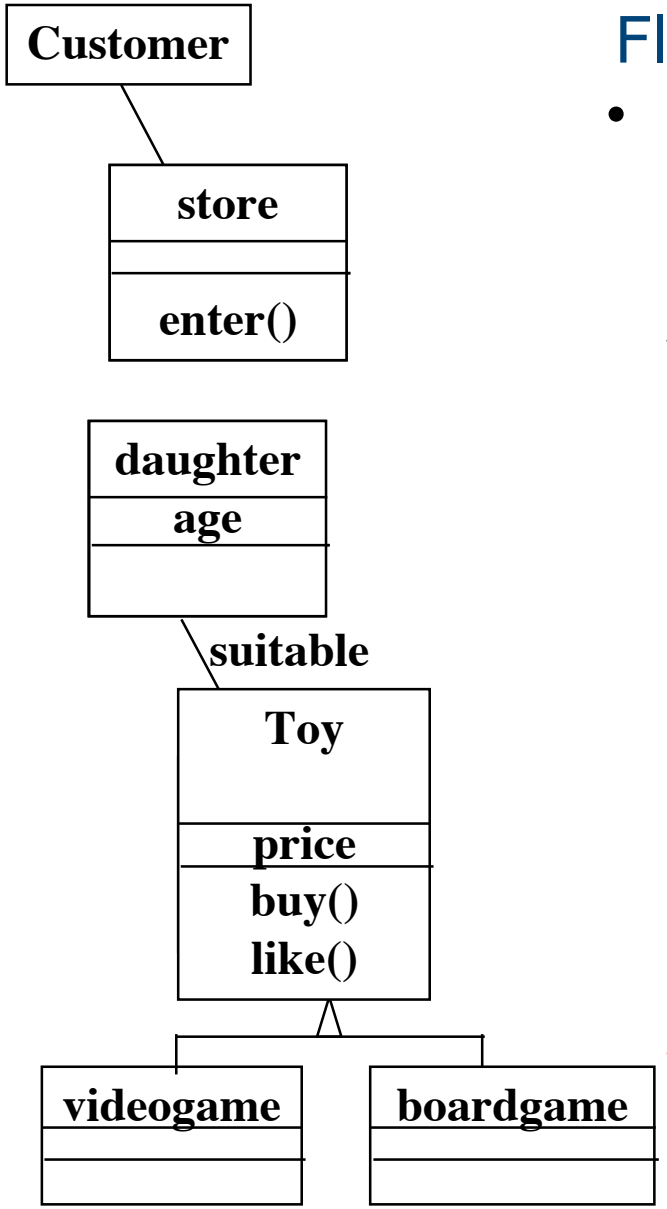
Flow of Events:

- The customer enters the store to buy a toy.
- It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- He tries a videogame, which uses a data glove and a head-mounted display. He likes it.
- An assistant helps him.
- The suitability of the game depends on the age of the child.
- His daughter is only 3 years old.
- The assistant recommends another type of toy, namely the boardgame “Monopoly".

Mapping parts of speech to model components (Abbot's Technique)

Example	*Part of speech*	*Model component*
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	Constraint, class, association

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

customer enters store
buy toy
daughter
less than 50
videogame

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

depends
age
type of toy
boardgame

Ways to find Objects

- Syntactical investigation with Abbot's technique:
  - Flow of events in use cases
  - Problem statement

- Use other knowledge sources:
  - Application knowledge: End users and experts know the abstractions of the application domain
  - Design knowledge: Abstractions in the solution domain
  - General world knowledge: Your generic knowledge and intution

Order of Activities for Object Identification

1. Formulate a few scenarios with help from an end user or application domain expert
2. Extract the use cases from the scenarios, with the help of an application domain expert
3. Then proceed in parallel with the following:
   - Analyse the flow of events in each use case using Abbot's textual analysis technique
   - Generate the UML class diagram.

Steps in Generating Class Diagrams

- Class identification (textual analysis, domain experts)
- Identification of attributes and operations (sometimes before the classes are found!)
- Identification of associations between classes
- Identification of multiplicities
- Identification of roles
- Identification of inheritance

Who uses Class Diagrams?

- Purpose of class diagrams
  - The description of the static properties of a system
- The main users of class diagrams:
  - The application domain expert
    - uses class diagrams to model the application domain (including taxonomies)
      - during requirements elicitation and analysis
  - The developer
    - uses class diagrams during the development of a system
      - during analysis, system design, object design and implementation.

Who does not use Class Diagrams?

- The client and the end user are often not interested in class diagrams
  - Clients usually focus more on project management issues
  - End users usually focus on the functionality of the system.

Developers have different Views on Class Diagrams

- According to the development activity, a developer plays different roles:
  - Analyst
  - System Designer
  - Object Designer
  - Implementor

- Each of these roles has a different view about the class diagram (the object model).

The View of the Analyst

- The analyst is interested
  - in application classes: The associations between classes are relationships between abstractions in the application domain
  - operations and attributes of the application classes (difference to E/R models!)
- The analyst uses inheritance in the model to reflect the taxonomies in the application domain
    - Taxonomy: An is-a-hierarchy of abstractions in an application domain
- The analyst is not interested
  - in the exact signature of operations
  - in solution domain classes

The View of the Designer

- The designer focuses on the solution of the problem, that is, the solution domain
- The associations between classes are now references (pointers) between classes in the application or solution domain
- An important design task is the specification of interfaces:
  - The designer describes the interface of classes and the interface of subsystems
  - Subsystems originate from modules (term often used during analysis):
    - Module: a collection of classes
    - Subsystem: a collection of classes with an interface
- Subsystems are modeled in UML with a package.

Goals of the Designer

- The most important design goals for the designer are design usability and design reusability
- Design usability: the interfaces are usable from as many classes as possible within in the system
- Design reusability: The interfaces are designed in a way, that they can also be reused by other (future) software systems
  - => Class libraries
  - => Frameworks
  - => Design patterns.

The View of the Implementor

- Class implementor
  - Must realize the interface of a class in a programming language
  - Interested in appropriate data structures (for the attributes) and algorithms (for the operations)
- Class extender
  - Interested in how to extend a class to solve a new problem or to adapt to a change in the application domain
- Class user
  - The class user is interested in the signatures of the class operations and conditions, under which they can be invoked
  - The class user is not interested in the implementation of the class.

Why do we distinguish different Users of Class Diagrams?

- Models often don't distinguish between application classes and solution classes
  - Reason: Modeling languages like UML allow the use of both types of classes in the same model
    - "address book", "array"
  - Preferred: No solution classes in the analysis model
- Many systems don't distinguish between the specification and the implementation of a class
  - Reason: Object-oriented programming languages allow the simultaneous use of specification and implementation of a class
  - Preferred: We distinguish between analysis model and object design model. The analysis design model does not contain any implementation specification.

Analysis model vs. object design model

- The analysis model is constructed during the analysis phase
  - Main stake holders: End user, customer, analyst
  - The class diagrams contains only application domain classes

- The object design model (sometimes also called specification model) is created during the object design phase
  - Main stake holders: class specifiers, class implementors, class users and class extenders
  - The class diagrams contain application domain as well as solution domain classes.

Analysis model vs object design model (2)

- The analysis model is the basis for communication between analysts, application domain experts and end users.
- The object design model is the basis for communication between designers and implementors.

Summary

- System modeling
  - Functional model, object model, dynamic model
- From scenarios to use cases to objects
- Object modeling is the central activity
  - Class identification is a major activity of object modeling
  - Easy syntactic rules to find classes and objects
  - Abbot's Technique
- Analysts, designers and implementors have different modeling needs
- There are three types of implementors with different roles during
  - Class user, class implementor, class extender.