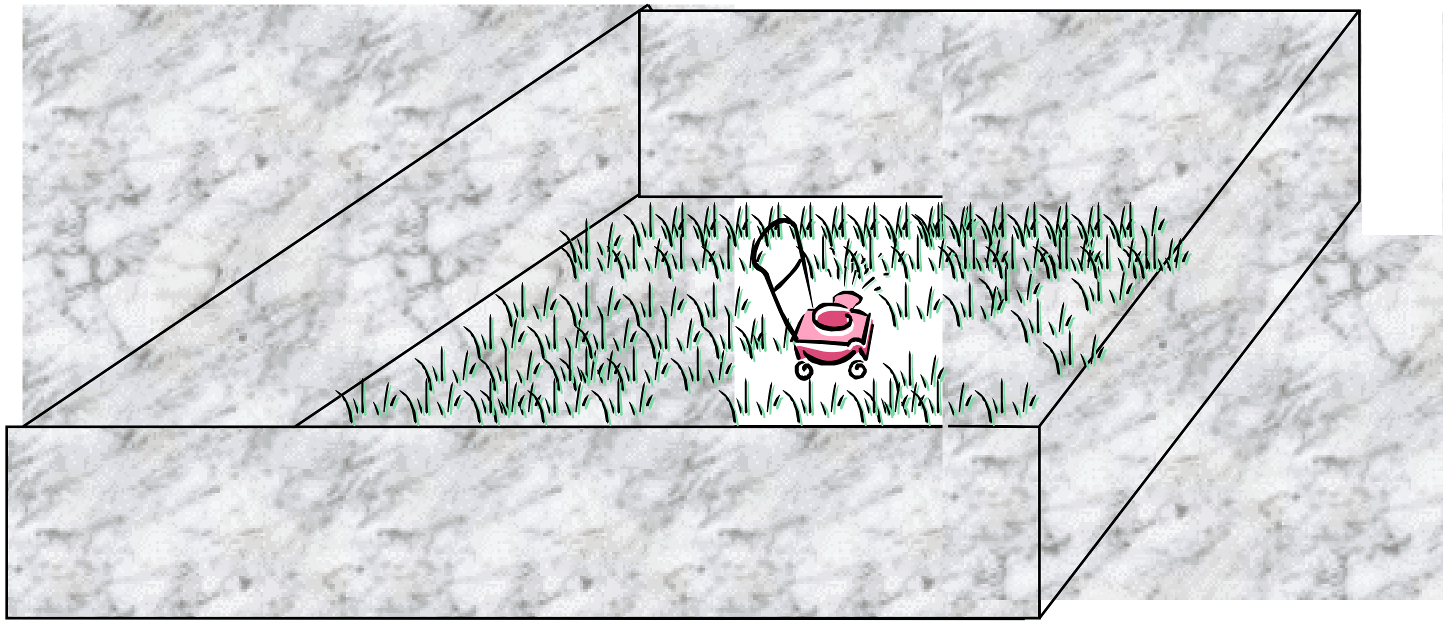
TUM

Requirements Elicitation

Bernd Brügge
Technische Universität München
Lehrstuhl für Angewandte Softwaretechnik

8 November 2001

What is this?



Location: Hochschule für Musik und Theater, Arcisstraße 12

Question: How do you mow the lawn?

Lesson: Find the functionality first, then the objects

Where are we right now?

- **Three ways to deal with complexity:**
  - Abstraction
  - Decomposition (Technique: Divide and conquer)
  - Hierarchy (Technique: Layering)
- **Two ways to deal with decomposition:**
  - Object-orientation and functional decomposition
  - Functional decompostion leads to unmaintainable code
  - Depending on the purpose of the system, different objects can be found
- **What is the right way?**
  - Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).
- **What activities and models are needed?**
  - This leads us to the software lifecycle we use in this class

Software Lifecycle Definition

- **Software lifecycle:**
  - Set of **activities** and their relationships to each other to support the development of a software system

- **Typical Lifecycle questions:**
  - Which activities should I select for the software project?
  - What are the dependencies between activities?
  - How should I schedule the activities?
  - What is the result of an activity

Example: Selection of Software Lifecycle Activities for a specific project

The Hacker knows only one activitity

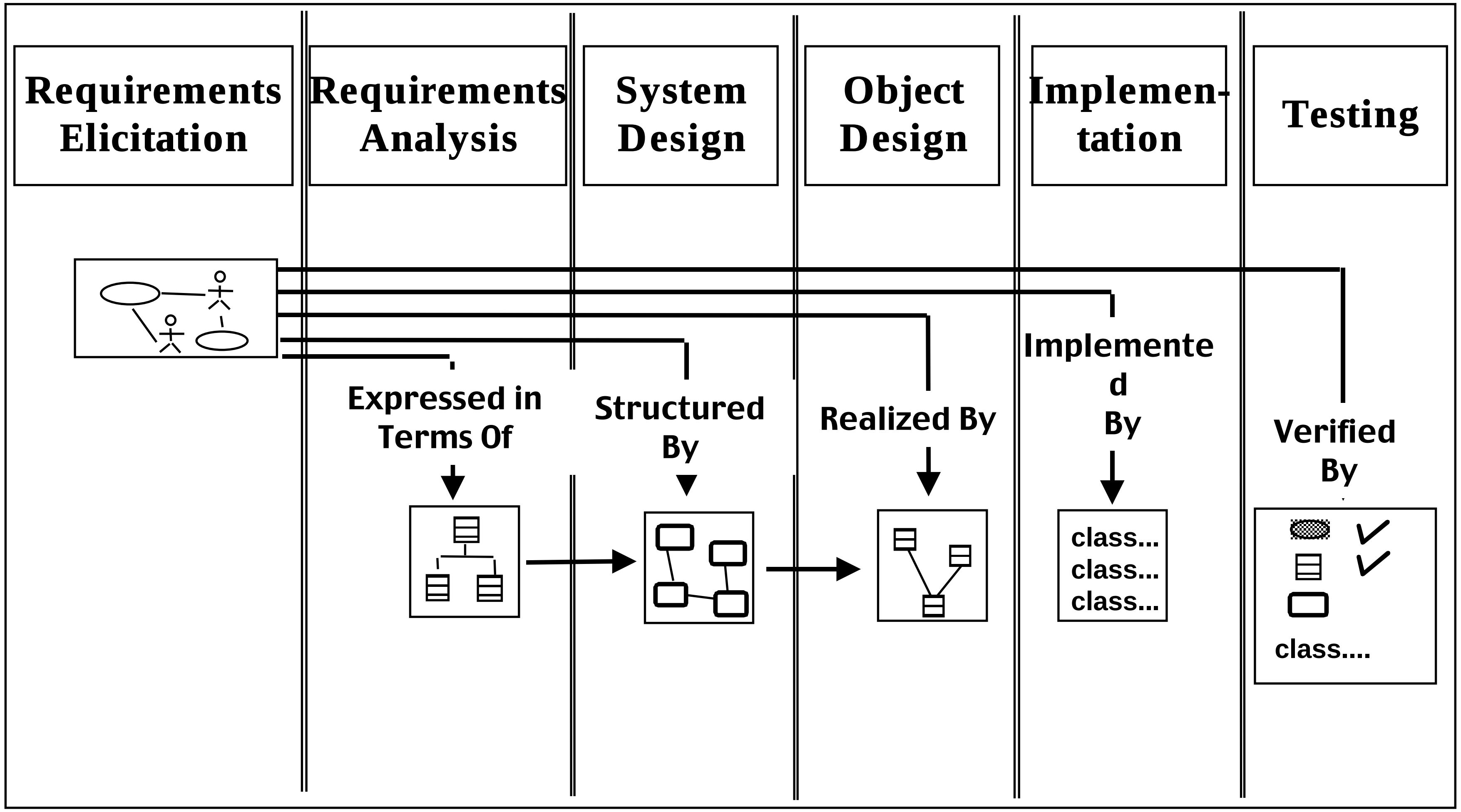
Implemen-tation

Activities used this lecture

Requirements Elicitation	**Analysis**	**System Design**	**Object Design**	**Implemen-tation**	**Testing**

Each activity produces one or more models

Software Lifecycle Activities

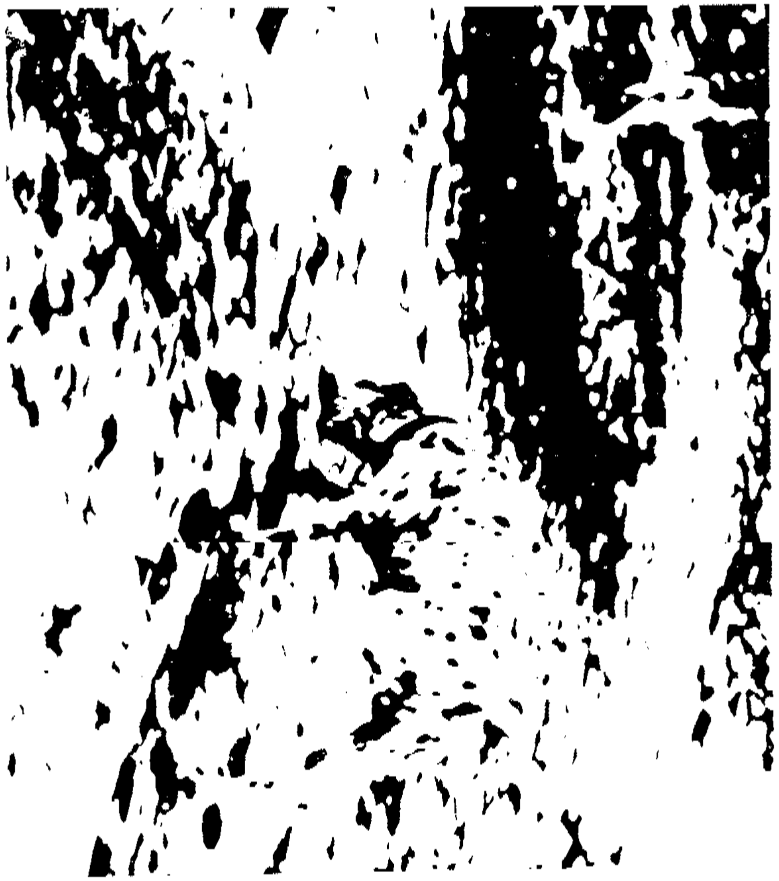


First Step in establishing the Requirements: System Identification

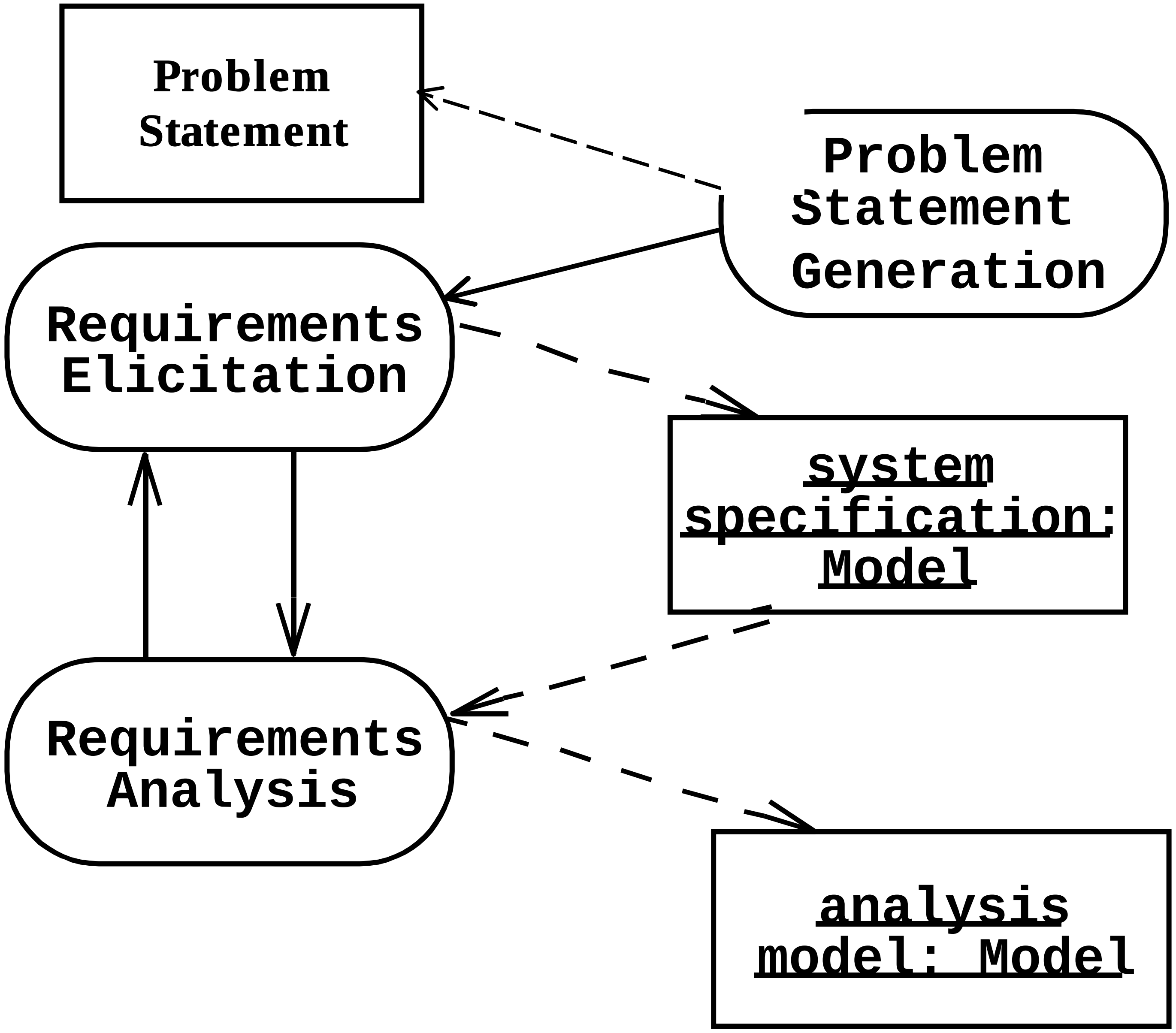
- **The development of a system is not just done by taking a snapshot of a scene (domain)**
- **Two questions need to be answered:**
  - How can we identify the purpose of a system?
  - Crucial is the definition of the system boundary: What is inside, what is outside the system?
- **These two questions are answered in the requirements process**
- **The requirements process consists of two activities:**
  - Requirements Elicitation:
    - Definition of the system in terms understood by the customer ("Problem Description")
  - Requirements Analysis:
    - Technical specification of the system in terms understood by the developer ("Problem Specification")

Defining the System Boundary is often difficult

What do you see here?



Products of Requirements Process (Activity Diagram)



Requirements Elicitation

- **Very challenging activity**
- **Requires collaboration of people with different backgrounds**
  - Users with application domain knowledge
  - Developer with solution domain knowledge (design knowledge, implementation knowledge)
- **Bridging the gap between user and developer:**
  - ***Scenarios:*** Example of the use of the system in terms of a series of interactions with between the user and the system
  - ***Use cases:*** Abstraction that describes a class of scenarios

System Specification vs Analysis Model

- **Both models focus on the requirements from the user's view of the system.**
- *System specification* **uses natural language (derived from the *problem statement*)**
- **The** *analysis model* **uses formal or semi-formal notation (for example, a graphical language like UML)**

- **The starting point is the problem statement**

Problem Statement

- **The problem statement is developed by the client as a description of the problem addressed by the system**
- **Other words for problem statement:**
  - Statement of Work
- **A good problem statement describes**
  - The current situation
  - The functionality the new system should support
  - The environment in which the system will be deployed
  - Deliverables expected by the client
  - Delivery dates
  - A set of acceptance criteria

Ingredients of a Problem Statement

- **Current situation: The Problem to be solved**
- **Description of one or more scenarios**
- **Requirements**
  - Functional and Nonfunctional requirements
  - Constraints (“pseudo requirements”)
- **Project Schedule**
  - Major milestones that involve interaction with the client including deadline for delivery of the system
- **Target environment**
  - The environment in which the delivered system has to perform a specified set of system tests
- **Client Acceptance Criteria**
  - Criteria for the system tests

Examples of Problem Statements

- **TRAMP:**
  - http://tramp.globalse.org/doc/PS/problem_statement_doc.pdf
  - The problem statement for the Softwaretechnik Praktikum Global Software Engineering
- **ARENA Project:**
  - http://tramp.globalse.org/doc/presentations/ARENA_Problem_Statement.pdf
  - The problem statement associated with this lecture on Software Engineering
  - Posted by tomorrow morning
- **Arena is not (yet) an acronym**
  - Challenge: Find a good acronym for ARENA

Current Situation: The Problem to be solved

- **There is a problem in the current situation**
  - Examples:
    - The response time when playing letter-chess is far too slow.
    - I want to play Go, but cannot find players on my level.
- **What has changed? Why can address the problem now?**
  - There has been a change, either in the application domain or in the solution domain
  - ***Change in the application domain***
    - A new function (business process) is introduced into the business
    - Example: We can play highly interactive games with remote people
  - ***Change in the solution domain***
    - A new solution (technology enabler) has appeared
    - Example: The internet allows the creation of virtual communities.

ARENA: The Problem

- **The Internet has enabled virtual communities**
  - Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g people in a chat room or playing a multi player game) or long lived (e.g., subscribers to a mailing list).
- **Many multi-player computer games now include support for virtual communities.**
  - Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.
- **Currently each game company develops such community support in each individual game.**
  - Each company uses a different infrastructure, different concepts, and provides different levels of support.
- **This redundancy and inconsistency leads to problems:**
  - High learning curve for players joining a new community,
  - Game companies need to develop the support from scratch
  - Advertisers need to contact each individual community separately.

ARENA: The Objectives

- **Provide a generic infrastructure for operating an arena to**
  - Support virtual game communities.
  - Register new games
  - Register new players
  - Organize tournaments
  - Keeping track of the players scores.
- **Provide a framework for tournament organizers**
  - to customize the number and sequence of matchers and the accumulation of expert rating points.
- **Provide a framework for game developers**
  - for developing new games, or for adapting existing games into the ARENA framework.
- **Provide an infrastructure for advertisers.**

Types of Requirements

- **Functional requirements:**
  - Describe the interactions between the system and its environment independent from implementation
  - Examples:
    - An ARENA operator should be able to define a new game.
- **Nonfunctional requirements:**
  - User visible aspects of the system not directly related to functional behavior.
  - Examples:
    - The response time must be less than 1 second
    - The ARENA server must be available 24 hours a day
- **Constraints ("Pseudo requirements"):**
  - Imposed by the client or the environment in which the system operates
    - The implementation language must be Java
    - ARENA must be able to dynamically interface to existing games provided by other game developers.

What is usually not in the Requirements?

- **System structure, implementation technology**
- **Development methodology**
- **Development environment**
- **Implementation language**
- **Reusability**

- **It is desirable that none of these above are constrained by the client. Fight for it!**

Requirements Validation

- **Requirements validation is a critical step in the development process, usually after requirements engineering or requirements analysis. Also at delivery (client acceptance test).**
- Requirements validation criteria:
  - Correctness:
    - **The requirements represent the client's view.**
  - Completeness:
    - **All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system**
  - Consistency:
    - **There are functional or nonfunctional requirements that contradict each other**
  - Realism:
    - **Requirements can be implemented and delivered**
  - Traceability:
    - **Each system function can be traced to a corresponding set of functional requirements**

Requirements Validation

- **Problem with requirements validation: Requirements change very fast during requirements elicitation.**
- **Tool support for managing requirements:**
  - Store requirements in a shared repository
  - Provide multi-user access
  - Automatically create a system specification document from the repository
  - Allow change management
  - Provide traceability throughout the project lifecycle
- **RequisitPro from Rational**
  - http://www.rational.com/products/reqpro/docs/datasheet.html
- **Request Tool (Allen Dutoit)**
  - Tomorrow's tutorial (November 9)

Types of Requirements Elicitation

- **Greenfield Engineering**
  - Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client
  - Triggered by user needs
  - **Example:** Develop a game from scratch: Asteroids
- **Re-engineering**
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
  - **Example:** Reengineering an existing game
- **Interface Engineering**
  - Provide the services of an existing system in a new environment
  - Triggered by technology enabler or new market needs
  - **Example:** Interface to an existing game (Bumpers)

Scenarios

- **"A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carrol, 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:**
  - Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.
    - Example: Description of Letter-Chess
- **Visionary scenario:**
  - Used to describe 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.
- **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 (greenfield engineering)**
- **Don’t wait for information even if the system exists**
- **Engage in a dialectic approach (evolutionary, incremental engineering)**
  - 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

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 *questionnaires* 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 software contractor
  - Expect resistance and try to overcome it

Example: Accident Management System

- **What needs to be done to report a “Cat in a Tree” incident?**
- **What do you need to do if a person reports “Warehouse on Fire?”**
- **Who is involved in reporting an incident?**
- **What does the system do, if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?**
- **What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?**
- **Can the system cope with a simultaneous incident report “Warehouse on Fire?”**

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, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. 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 two paramedic units to the Incident site and sends their 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

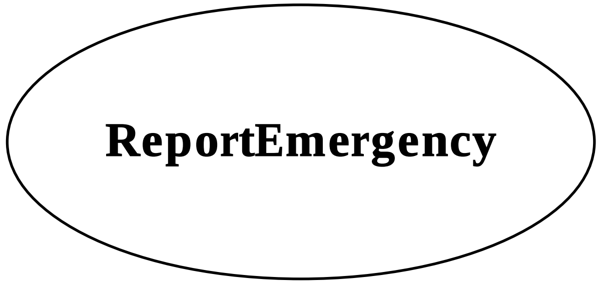
Next goal, after the scenarios are formulated:

- **Find all the use cases in the scenario that specifies all possible 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 Special Requirements (Constraints, Nonfunctional Requirements

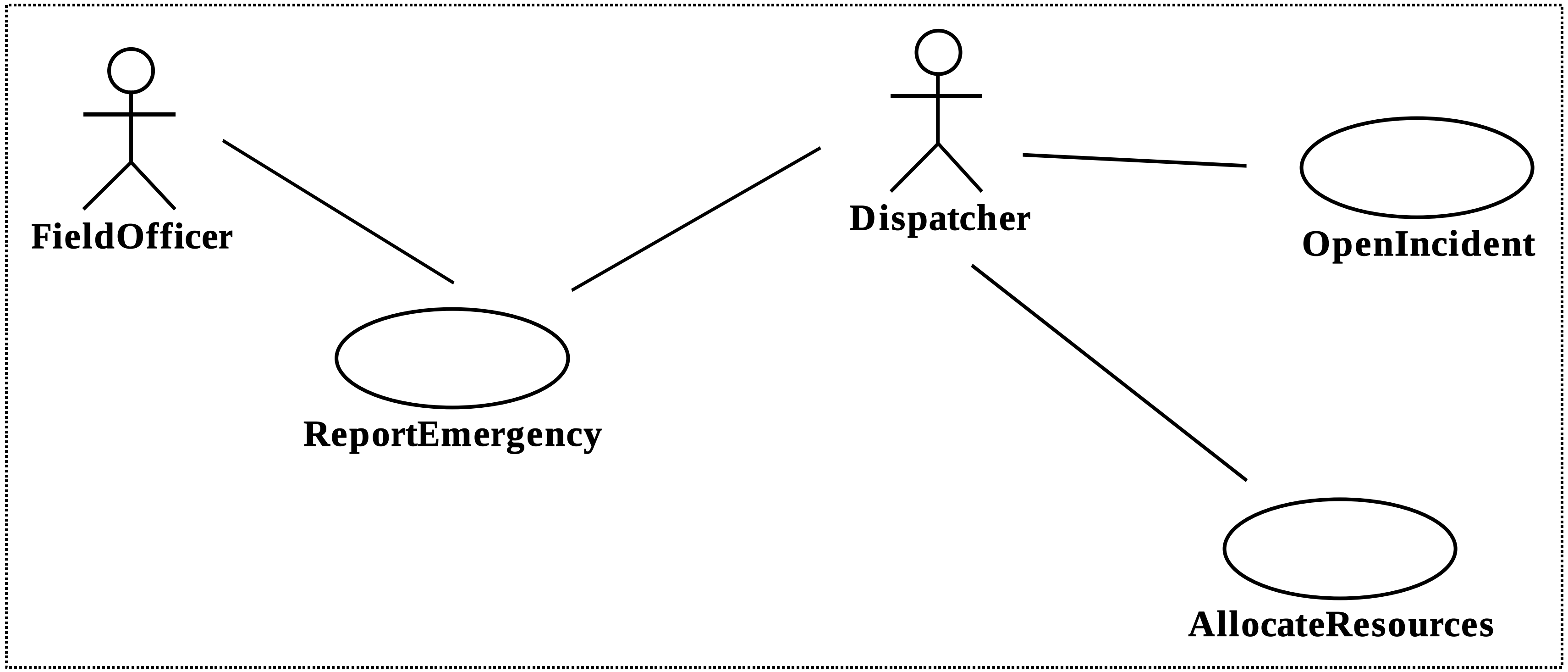
Use Cases

- **A use case is a flow of events in the system, including interaction with actors**
- **It is initiated by an actor**
- **Each use case has a name**
- **Each use case has a termination condition**
- **Graphical Notation: An oval with the name of the use case**



- ***Use Case Model:*** The set of all use cases specifying the complete functionality of the system

Example: Use Case Model for Incident Management



Heuristics: How do I 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 her terminal and the central is lost.
  - The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the 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

- **The** FieldOfficer **activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.**
- **The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the** Dispatcher **is notified.**
- **The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.**
- **The FieldOfficer receives the acknowledgment and the selected response.**

Another Use Case Example: Allocate a Resource

- **Actors:**
  - ***Field Supervisor:*** This is the official at the emergency site....
  - ***Resource Allocator:*** The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system. ...
  - ***Dispatcher:*** A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.
  - ***Field Officer:*** Reports accidents from the Field

Another Use Case Example: Allocate a Resource

- *Use case name*: **AllocateResources**
- ***Participating Actors:***
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
  - Resource Allocator
  - Field Supervisor
- ***Entry Condition***
  - The Resource Allocator has selected an available resource.
  - The resource is currently not allocated
- ***Flow of Events***
  - The Resource Allocator selects an Emergency Incident.
  - The Resource is committed to the Emergency Incident.
- ***Exit Condition***
  - The use case terminates when the resource is committed.
  - The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.
- ***Special Requirements***
  - The Field Supervisor is responsible for managing the Resources

Order of steps when formulating use cases

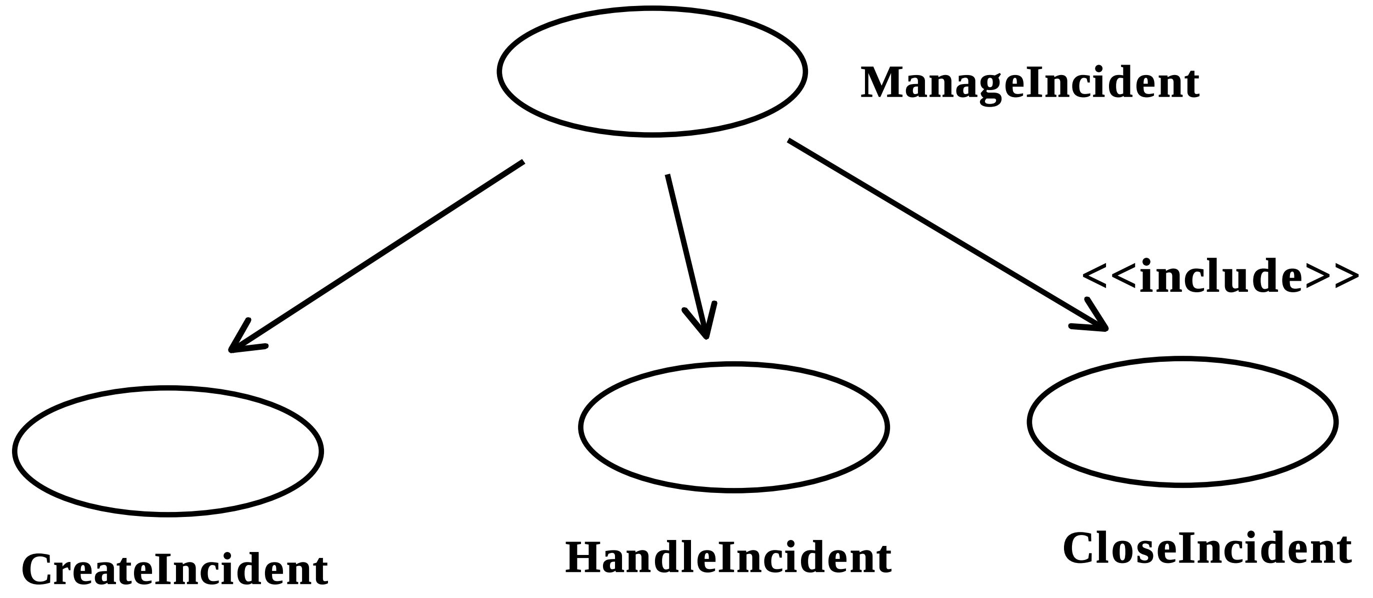
- **First step: name the use case**
  - Use case name: ReportEmergency
- **Second step: Find the actors**
  - Generalize the concrete names ("Bob") to participating actors ("Field officer")
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)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

Use Case Associations

- **A use case model consists of use cases and use case associations**
  - A use case association is a relationship between use cases
- **Important types of use case associations: Include, Extends, Generalization**
- **Include**
  - A use case uses another use case (“functional decomposition”)
- **Extends**
  - A use case extends another use case
- **Generalization**
  - An abstract use case has different specializations

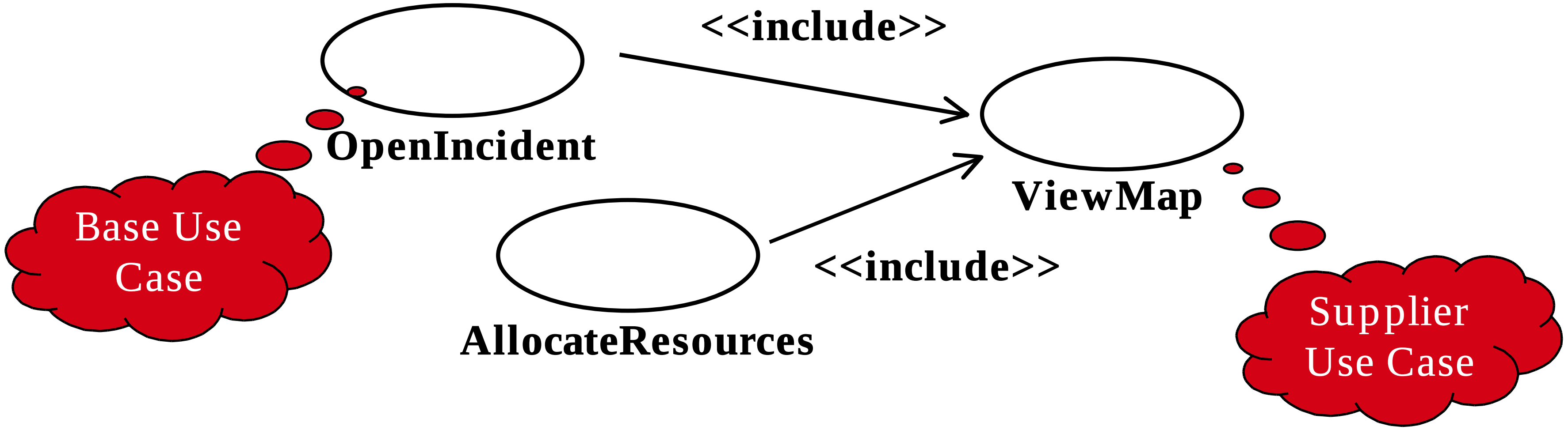
<<Include>>: Functional Decomposition

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



<<Include>>: Reuse of Existing Functionality

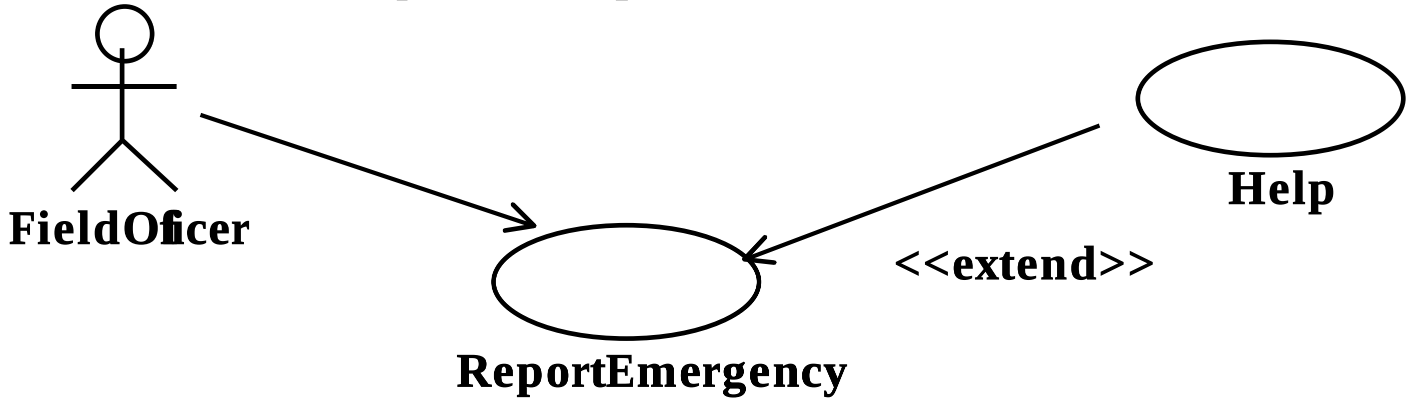
- **Problem:**
  - There are already existing functions. How can we *reuse* them?
- **Solution:**
  - The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B (“A delegates to B”)
- **Example:**
  - The use case “ViewMap” describes behavior that can be used by the use case “OpenIncident” (“ViewMap” is factored out)



- Note: The base case cannot exist alone. It is always called with the supplier use case

<Extend>> Association for Use Cases

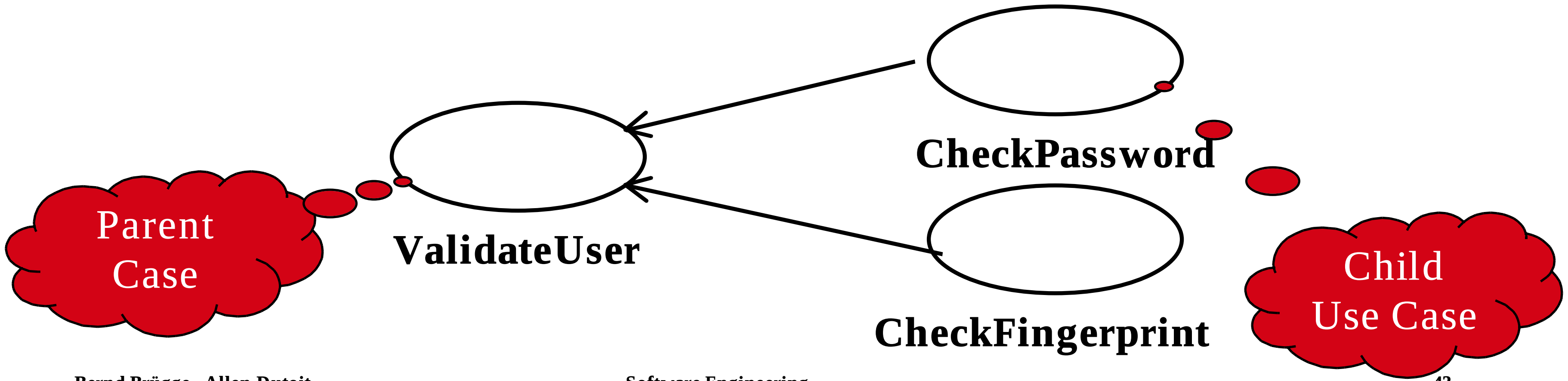
- **Problem:**
  - The functionality in the original problem statement needs to be extended.
- **Solution:**
  - An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.
- **Example:**
  - The use case "ReportEmergency" is complete by itself ,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



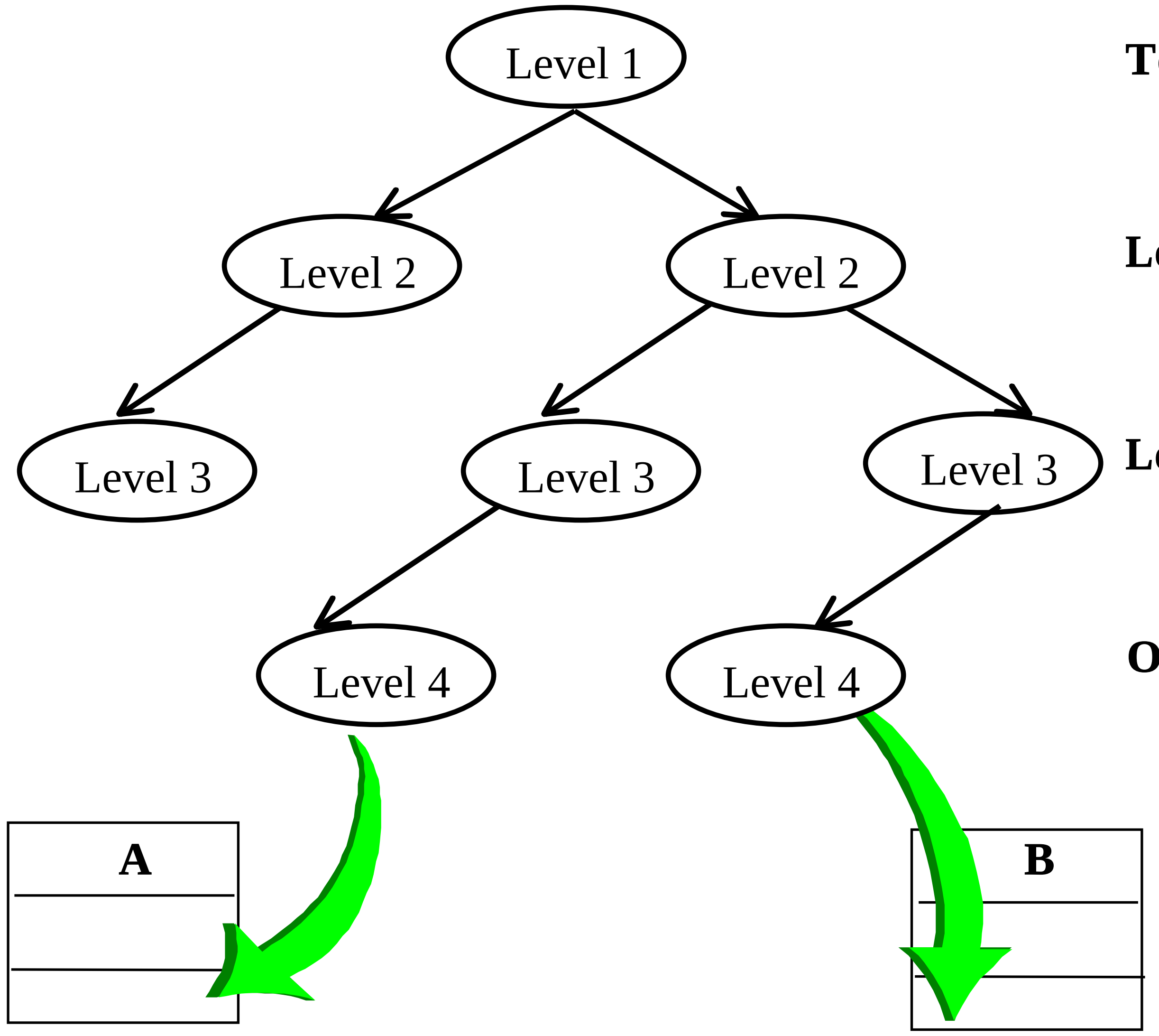
- Note: The base use case can be executed without the use case extension in extend assocations.

Generalization association in use cases

- **Problem:**
  - You have common behavior among use cases and want to factor this out.
- **Solution:**
  - The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- **Example:**
  - Consider the use case “ValidateUser”, responsible for verifying the identity of the user. The customer might require two realizations: “CheckPassword” and “CheckFingerprint”



From Use Cases to Objects



Top Level Use Case

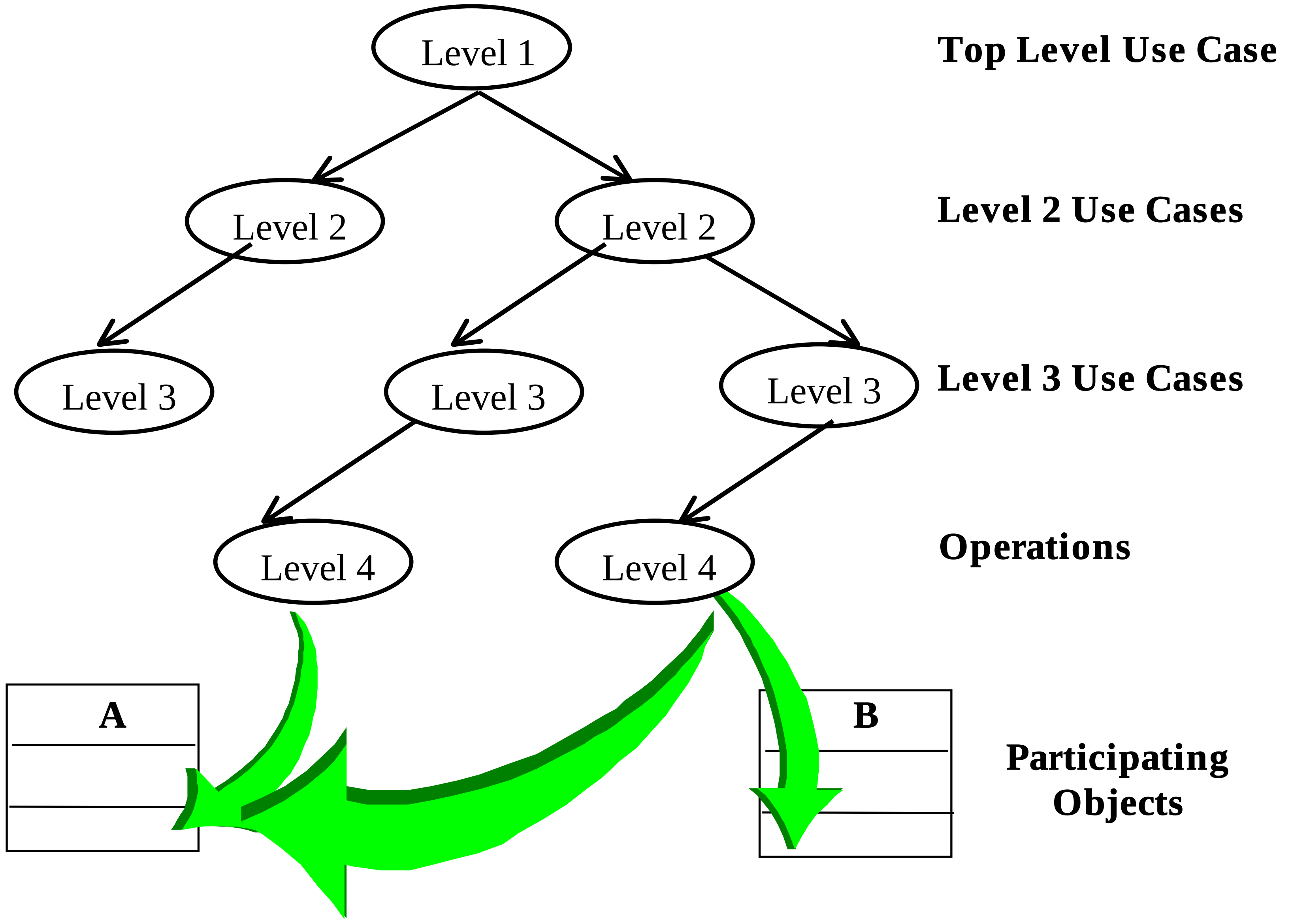
Level 2 Use Cases

Level 3 Use Cases

Operations

A and B are called Participating Objects

Use Cases can be used by more than one object



How to Specify 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

- **The requirements process consists of requirements elicitation and analysis.**
- **The requirements elicitation activity is different for:**
  - Greenfield Engineering, Reengineering, Interface Engineering
- **Scenarios:**
  - Great way to establish communication with client
  - Different types of scenarios: As-Is, visionary, evalution and training
  - Use cases: Abstraction of scenarios
- **Pure functional decomposition is bad:**
  - Leads to unmaintainable code
- **Pure object identification is bad:**
  - May lead to wrong objects, wrong attributes, wrong methods
- **The key to successful analysis:**
  - Start with use cases and then find the participating objects
  - If somebody asks "What is this?", do not answer right away. Return the question or observe the end user: "What is it used for?"