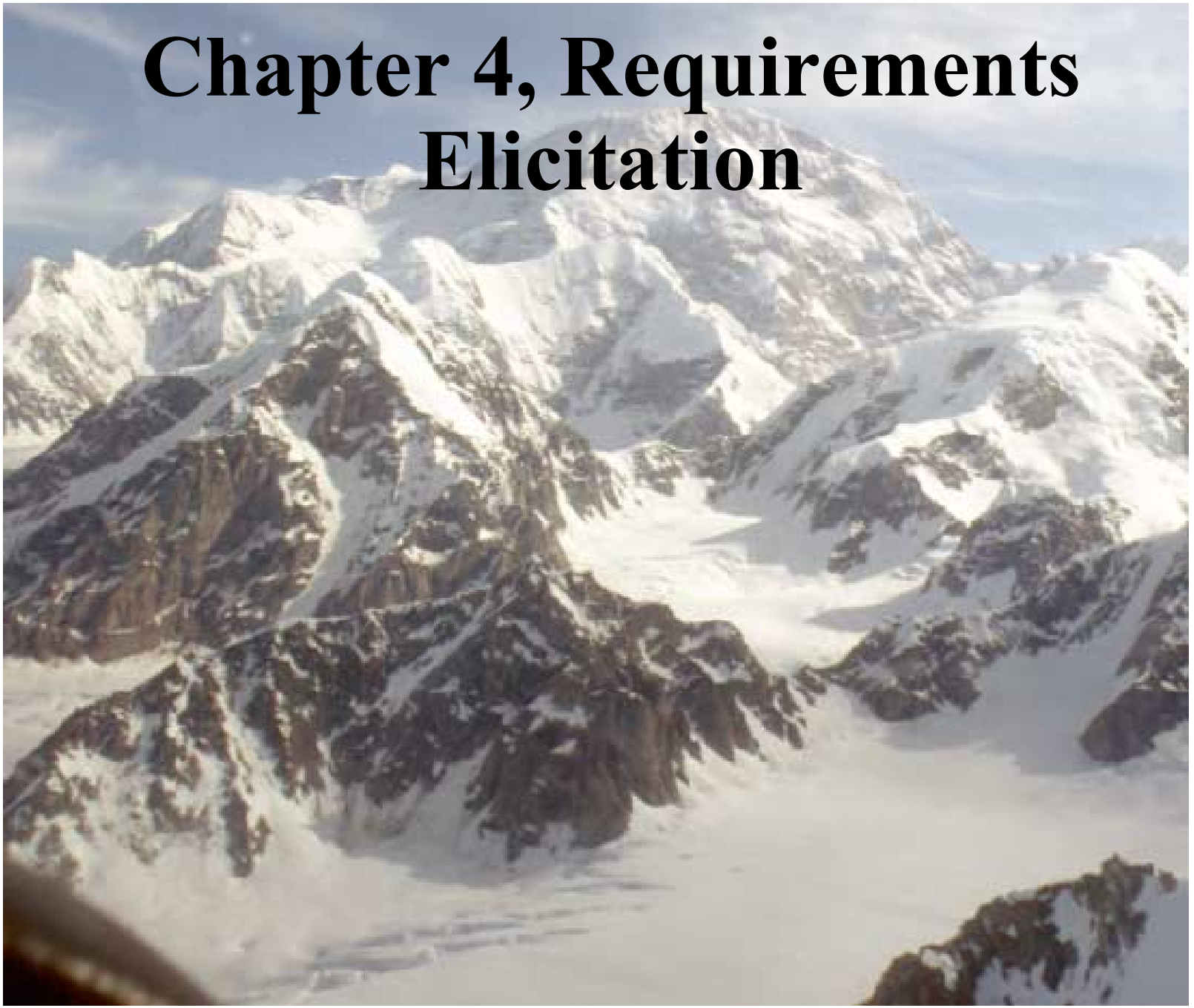
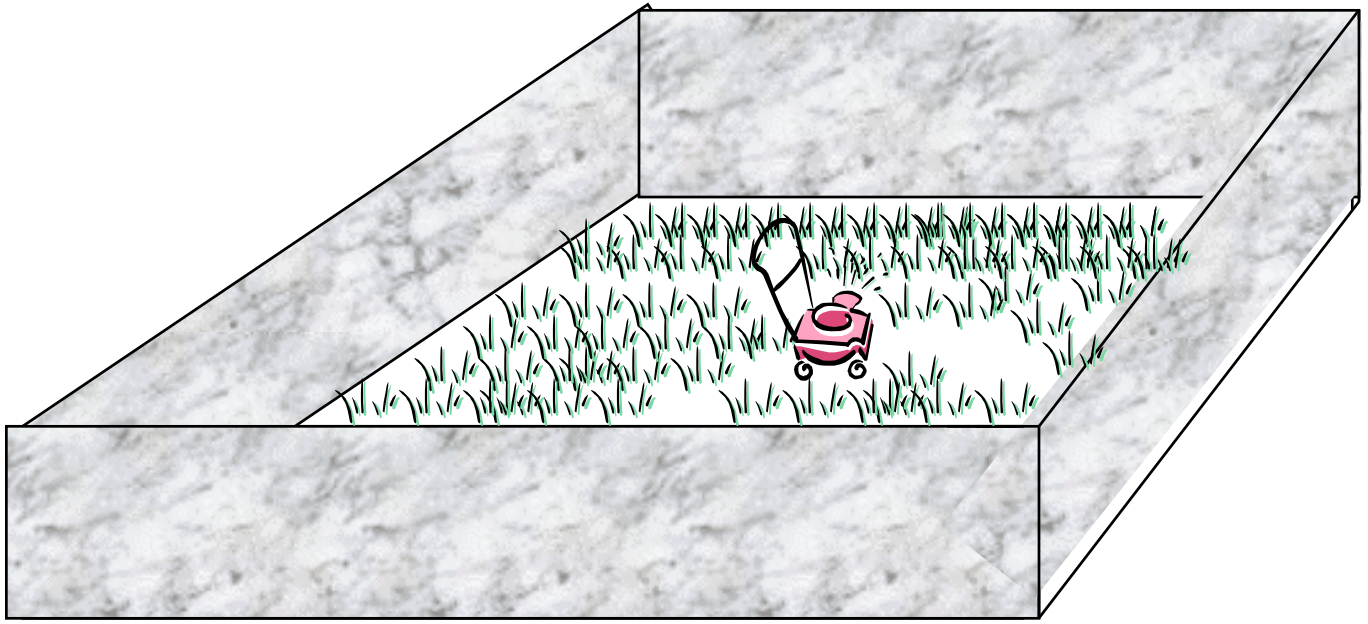
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 4, Requirements Elicitation



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?

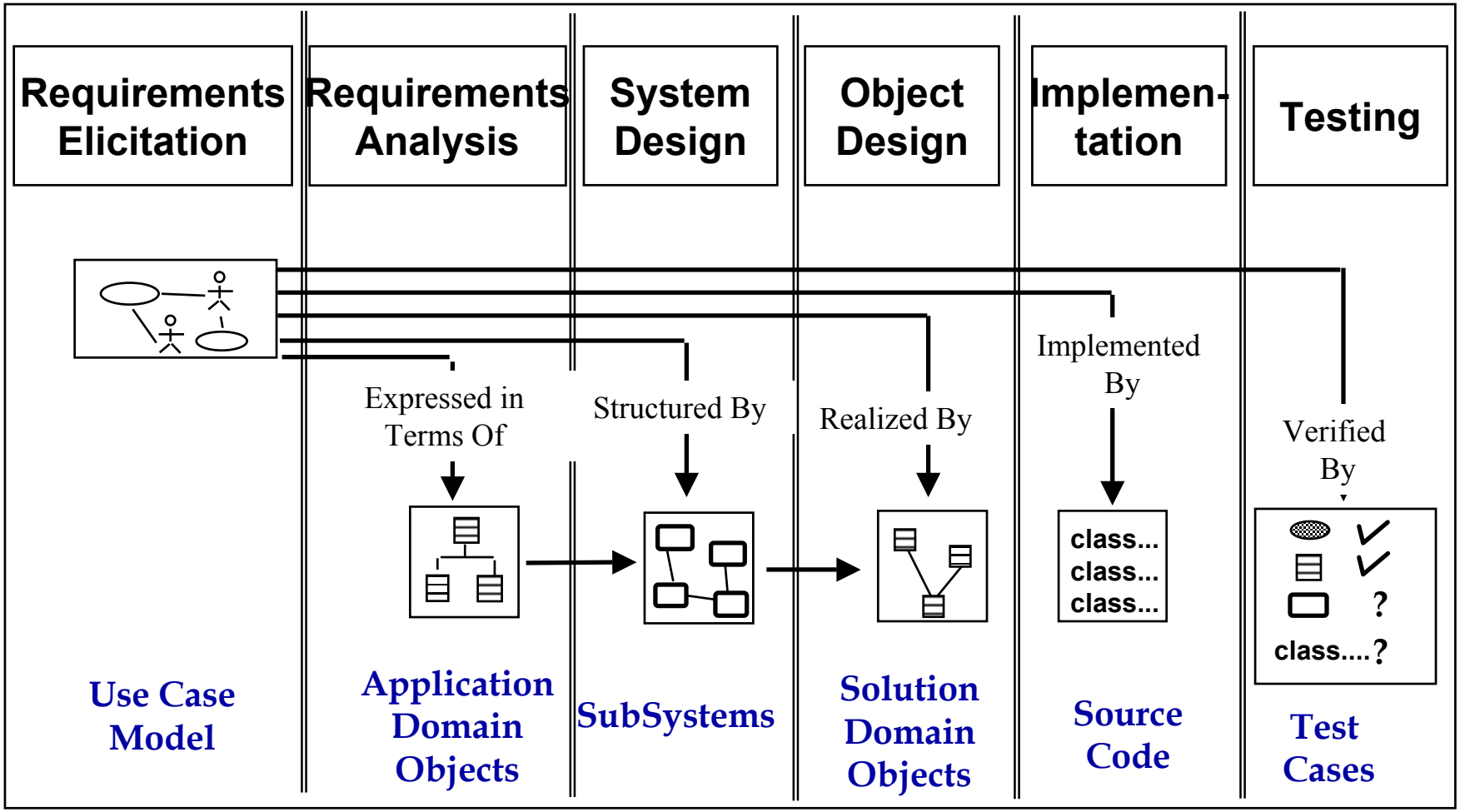
- Three ways to deal with complexity:
  - **Abstraction**
  - **Decomposition (Technique: Divide and conquer)**
  - **Hierarchy (Technique: Layering)**
- Two ways to deal with decomposition:
  - **Object-orientation and functional decomposition**
  - **Functional decomposition leads to unmaintainable 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

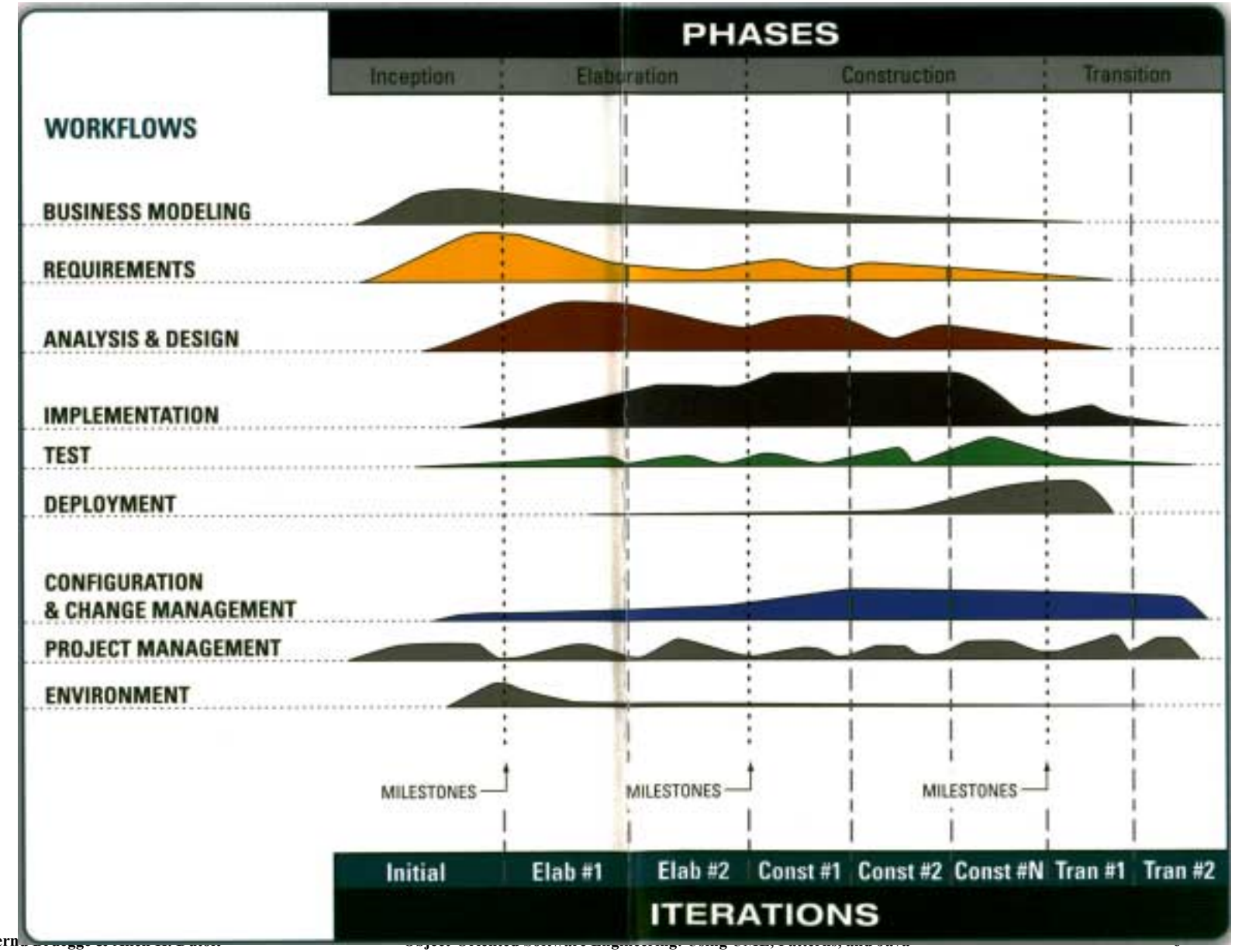
- Software lifecycle:
  - **Set of** activities **and their relationships to each other to support the development of a software system**

- 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**

Software Lifecycle Activities



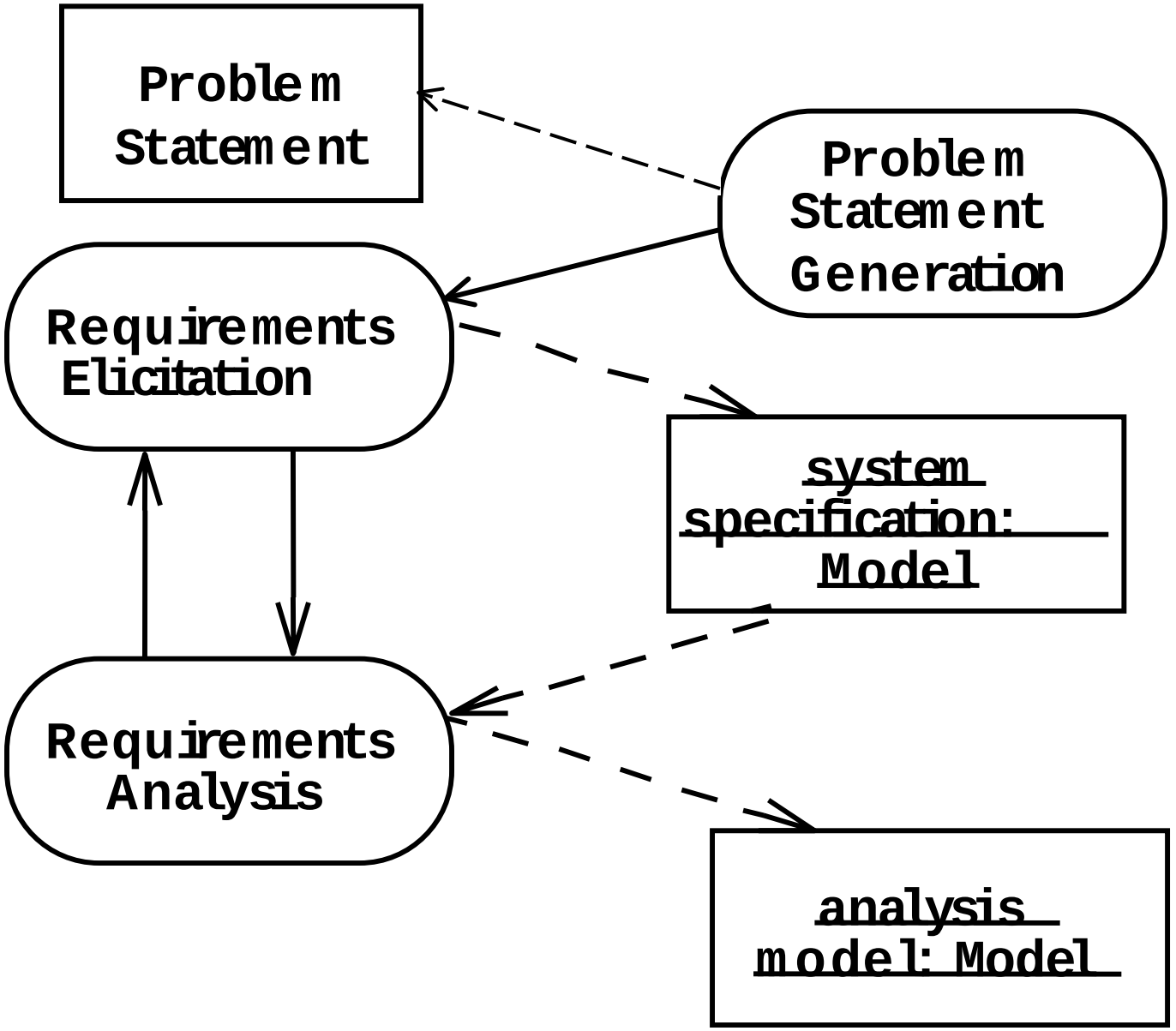
Rational Unified Process (RUP)



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")**

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**

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? How to address the changed problem?
  - **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.**

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**

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**

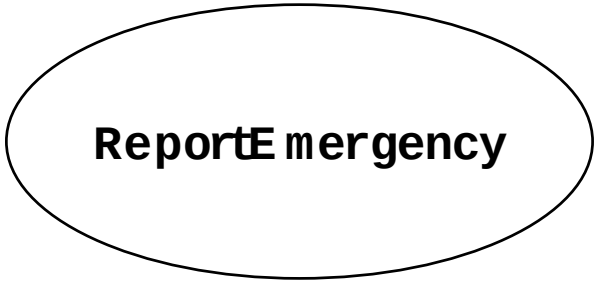
Next goal, after the scenarios are formulated:

- 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**

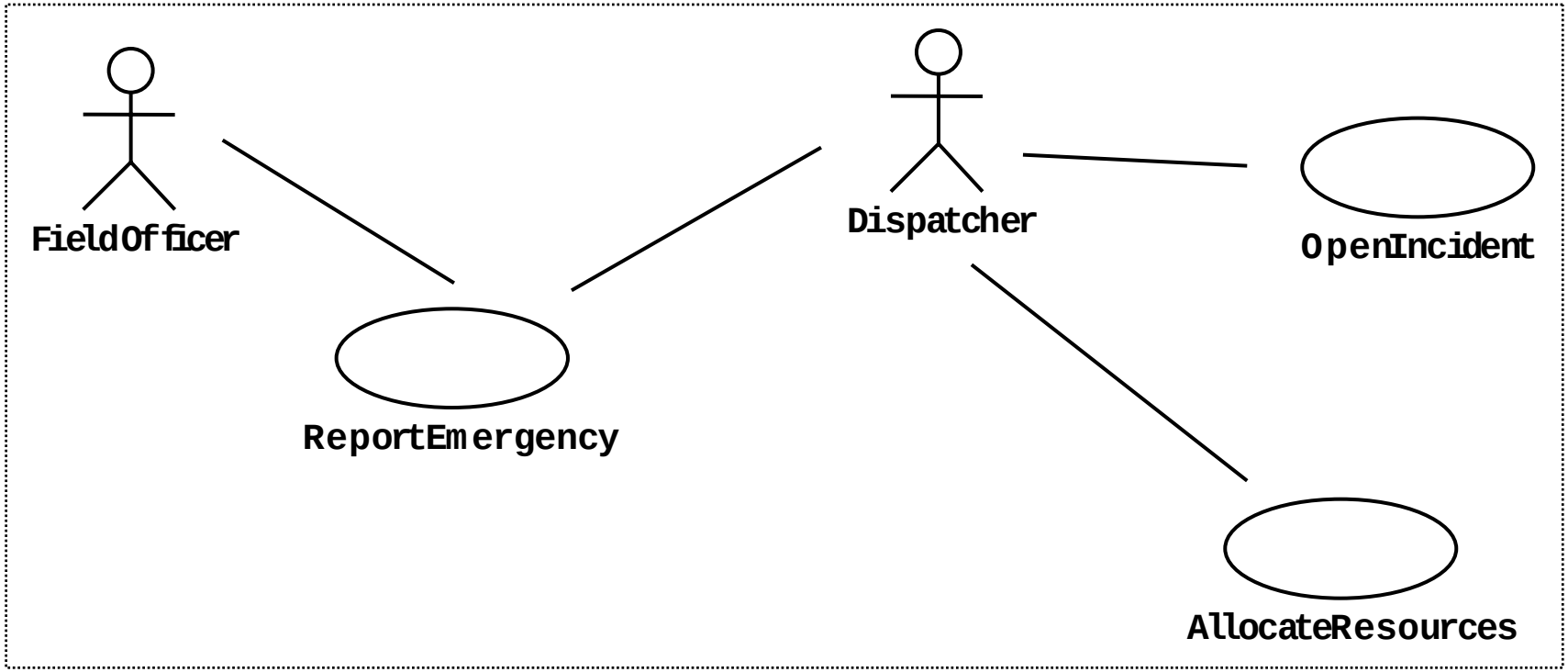
Use Cases

- 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)**

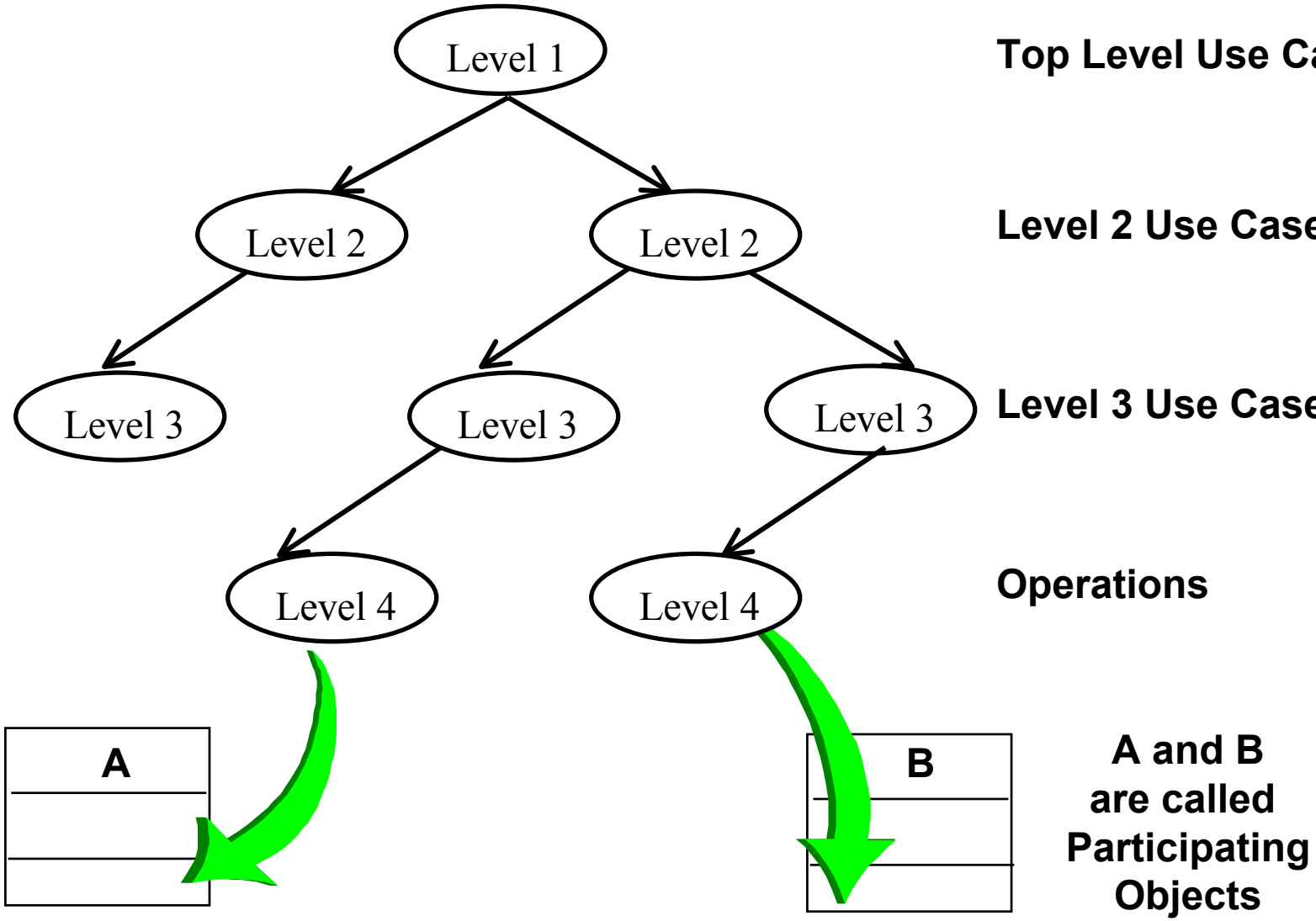
Order of steps when formulating use cases

- 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**

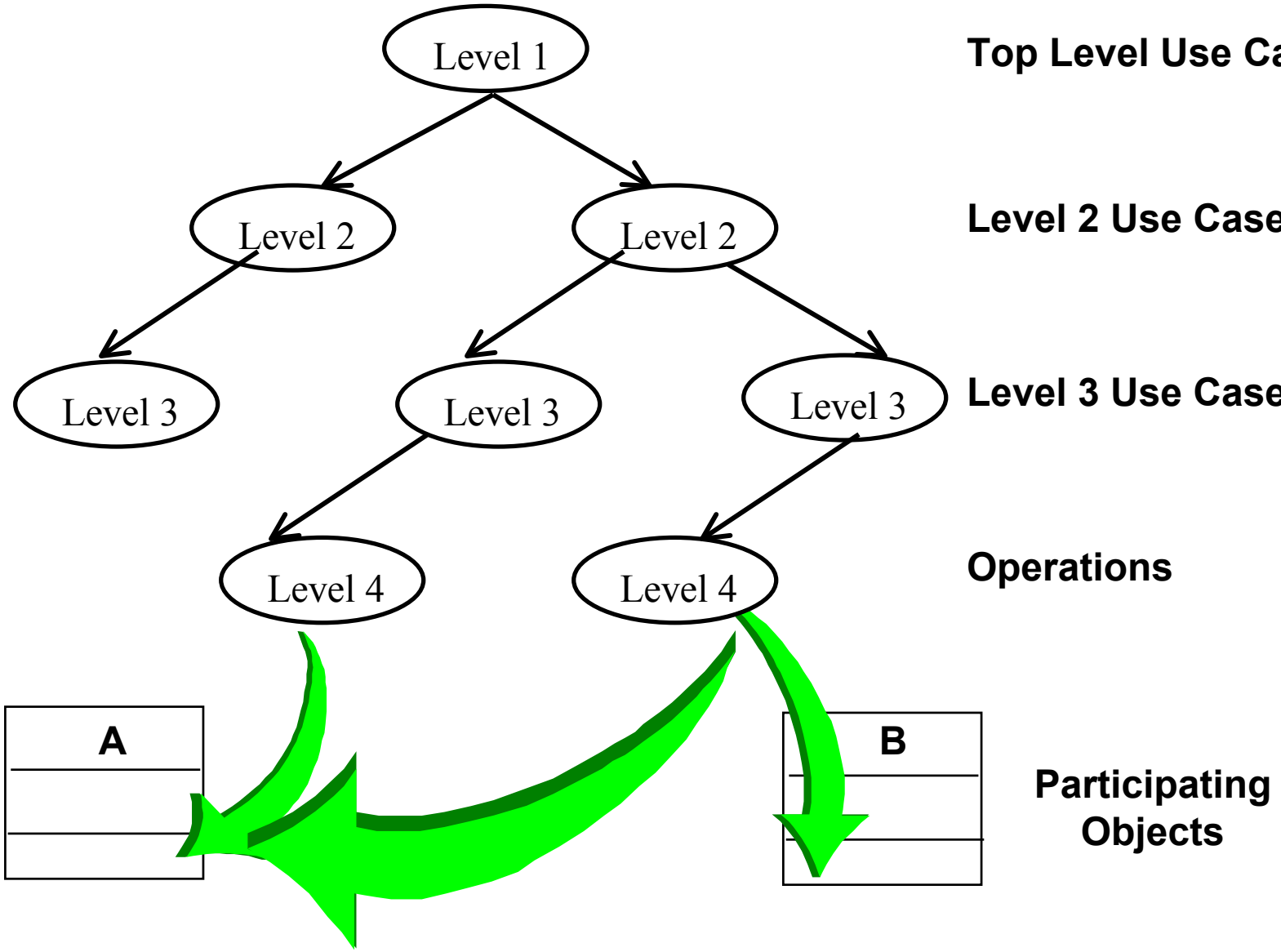
Use Case Associations

- 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**

From Use Cases to 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, evaluation 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?"**