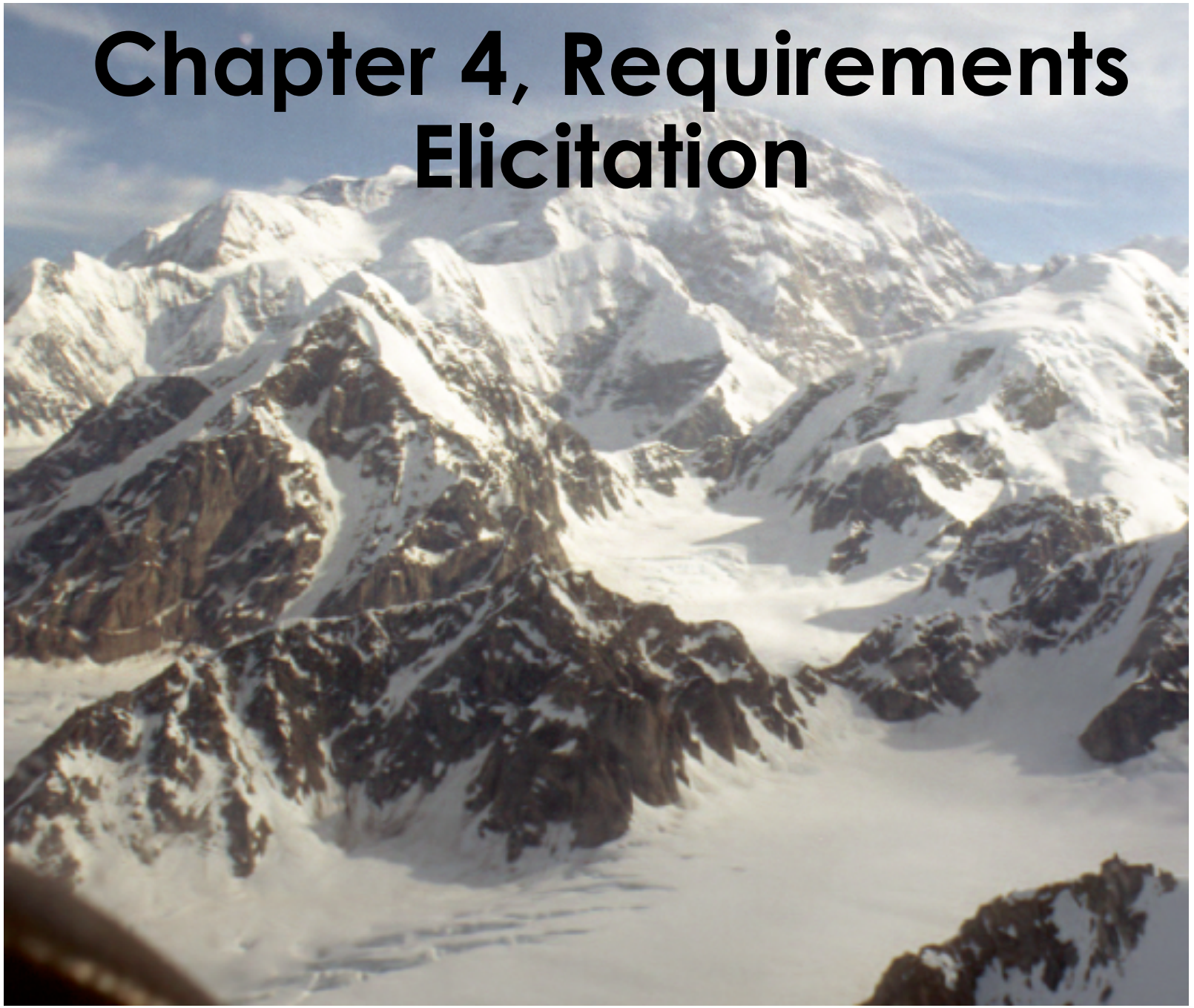
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 4, Requirements Elicitation



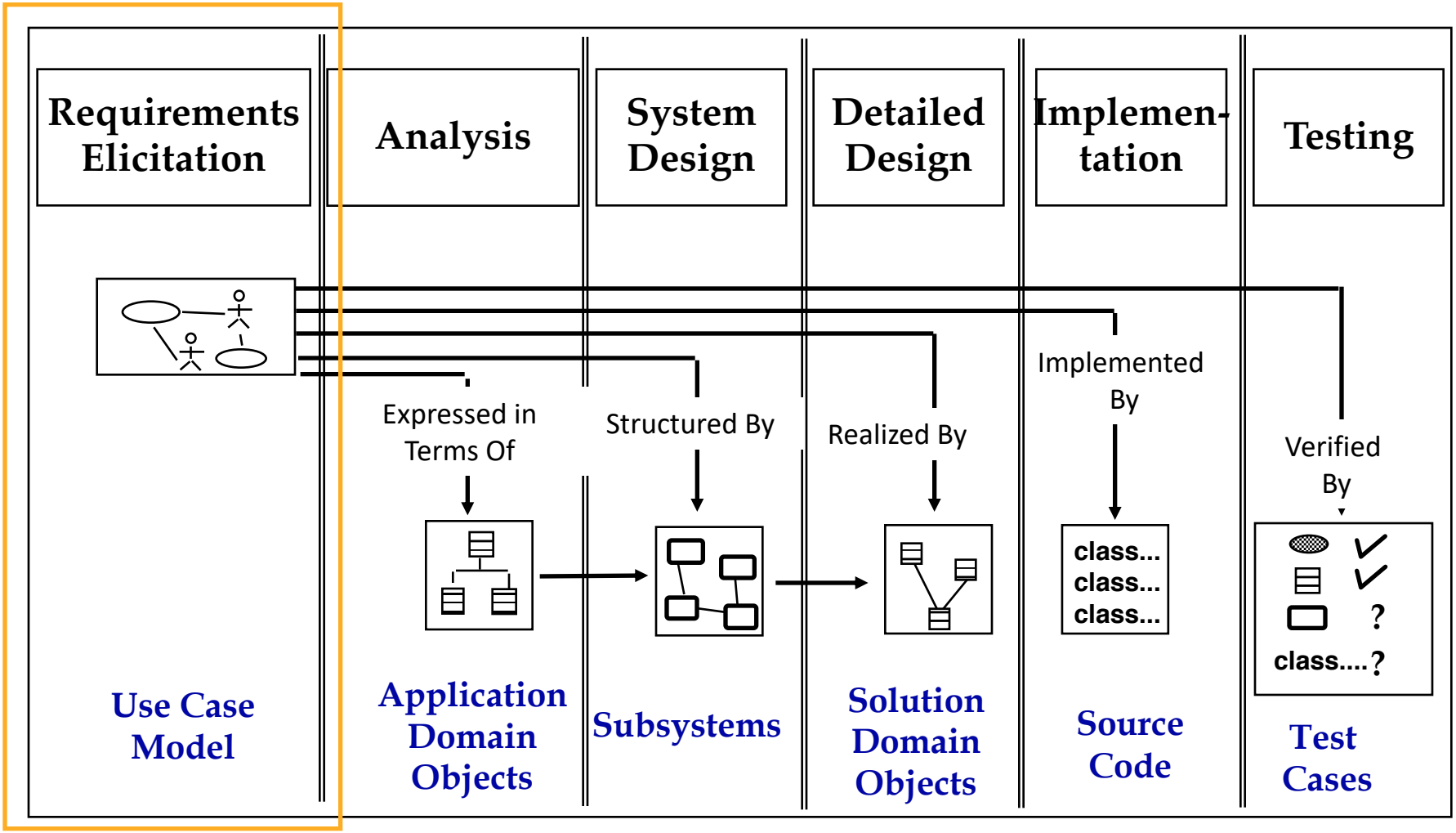
Outline

- Motivation: Software Lifecycle
- Requirements elicitation challenges
- Problem statement
- Requirements specification
  - Types of requirements
- Validating requirements
- Summary

Software Lifecycle Definition

- Software lifecycle
  - Models for the development of software
    - Set of activities and their dependency relationships to each other to support the development of a software system
    - Examples:
      - Analysis, Design, Implementation, Testing
- Typical Lifecycle questions:
  - Which activities should I select when I develop software?
  - What are the dependencies between activities?
  - How should I schedule the activities?

Software Lifecycle Activities



First step in identifying the Requirements: System identification

- Two questions need to be answered:
  1. How can we identify the purpose of a system?
  2. What is inside, what is outside the system?
- These two questions are answered during requirements elicitation and analysis
- Requirements elicitation:
  - Definition of the system in terms understood by the customer ("Requirements specification")
- Analysis:
  - Definition of the system in terms understood by the developer (Technical specification, "Analysis model")
- Requirements Process: Contains the activities Requirements Elicitation and Analysis.

Techniques to elicit Requirements

- Bridging the gap between end user and developer:
  - **Questionnaires:** Asking the end user a list of pre-selected questions
  - **Task Analysis:** Observing end users in their operational environment
  - **Scenarios:** Describe the use of the system as a series of interactions between a concrete end user and the system
  - **Use cases:** Abstractions that describe a class of scenarios.

Scenarios

- Scenario
  - A synthetic description of an event or series of actions and events.
  - A textual description of the usage of a system. The description is written from an end user's point of view.
  - A scenario can include text, video, pictures and story boards. It usually also contains details about the work place, social situations and resource constraints.

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.

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

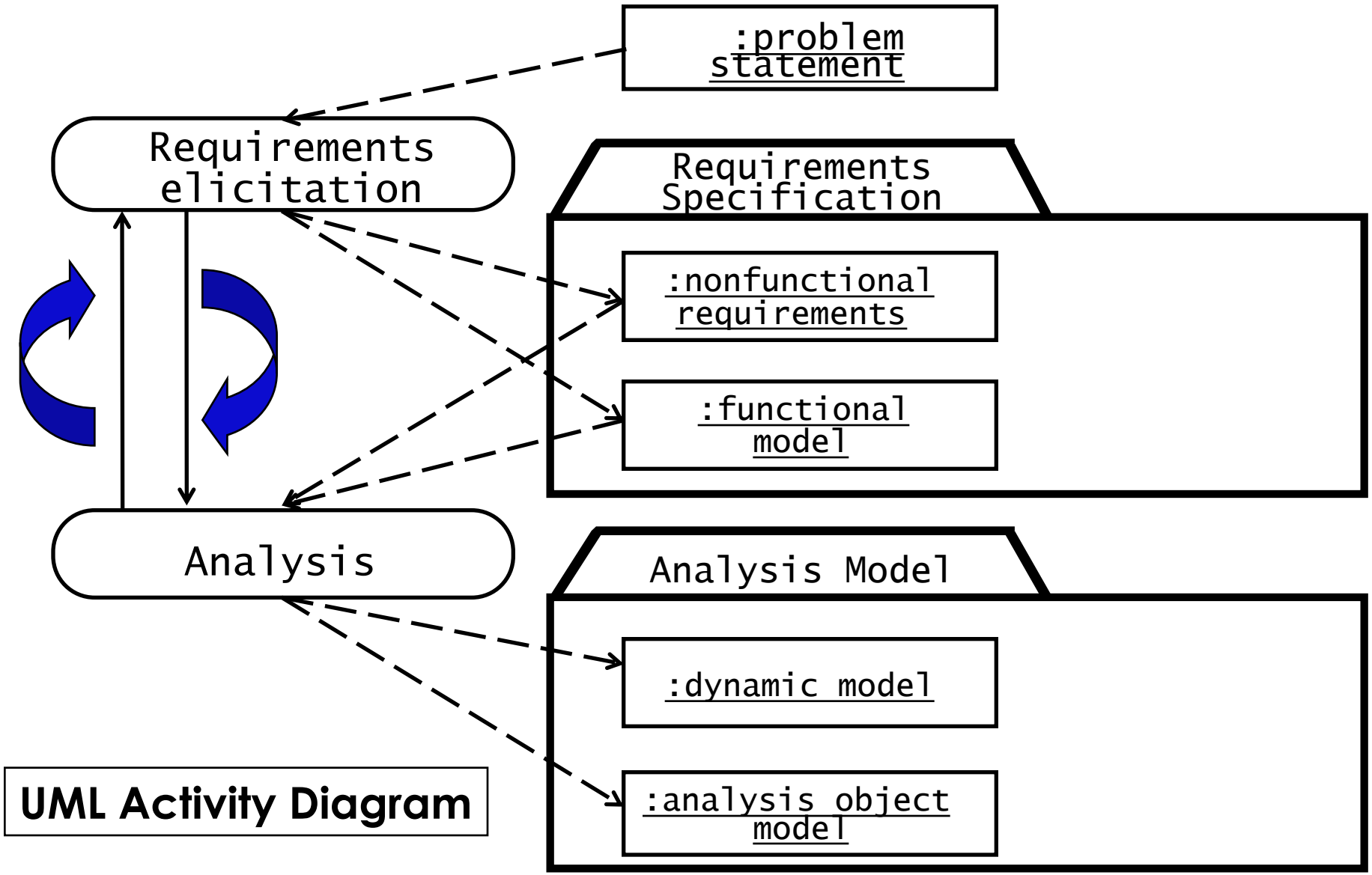
Heuristics for finding scenarios

- 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 *and* 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 client
  - Expect resistance and try to overcome it.

Requirements Elicitation: Difficulties and Challenges

- Communicate accurately about the domain and the system
  - People with different backgrounds must collaborate to bridge the gap between end users and developers
    - Client and end users have application domain knowledge
    - Developers have solution domain knowledge
- Identify an appropriate system (Defining the system boundary)
- Provide an unambiguous specification
- Leave out unintended features

Requirements Process



Requirements Specification vs Analysis Model

Both focus on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal or semi-formal notation
  - We use UML.

Types of Requirements

- Functional requirements
  - Describe the interactions between the system and its environment independent from the implementation

    "An operator must be able to define a new game. "
- Nonfunctional requirements
  - Aspects not directly related to functional behavior.

    "The response time must be less than 1 second"
- Constraints
  - Imposed by the client or the environment
    - "The implementation language must be Java "
  - Called "Pseudo requirements" in the text book.

Functional vs. Nonfunctional Requirements

Functional Requirements

- Describe user tasks that the system needs to support
- Phrased as actions
  - "Advertise a new league"
  - "Schedule tournament"
  - "Notify an interest group"

Nonfunctional Requirements

- Describe properties of the system or the domain
- Phrased as constraints or negative assertions
  - "All user inputs should be acknowledged within 1 second"
  - "A system crash should not result in data loss".

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
- Performance
  - Response time
  - Scalability
  - Throughput
  - Availability
- Supportability
  - Adaptability
  - Maintainability

Quality requirements

- Implementation
- Interface
- Operation
- Packaging
- Legal
  - Licensing (GPL, LGPL)
  - Certification
  - Regulation

Constraints or
Pseudo requirements

What should not be 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.

Prioritizing requirements

- High priority
  - Addressed during analysis, design, and implementation
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during analysis and design
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed only during analysis
  - Illustrates how the system is going to be used in the future with not yet available technology

Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 Constraints (“Pseudo requirements”)
   - 3.5 System models
     - 3.5.1 Scenarios
     - 3.5.2 Use case model
     - 3.5.3 Object model
       - 3.5.3.1 Data dictionary
       - 3.5.3.2 Class diagrams
     - 3.5.4 Dynamic models
     - 3.5.5 User interfae
4. Glossary

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.

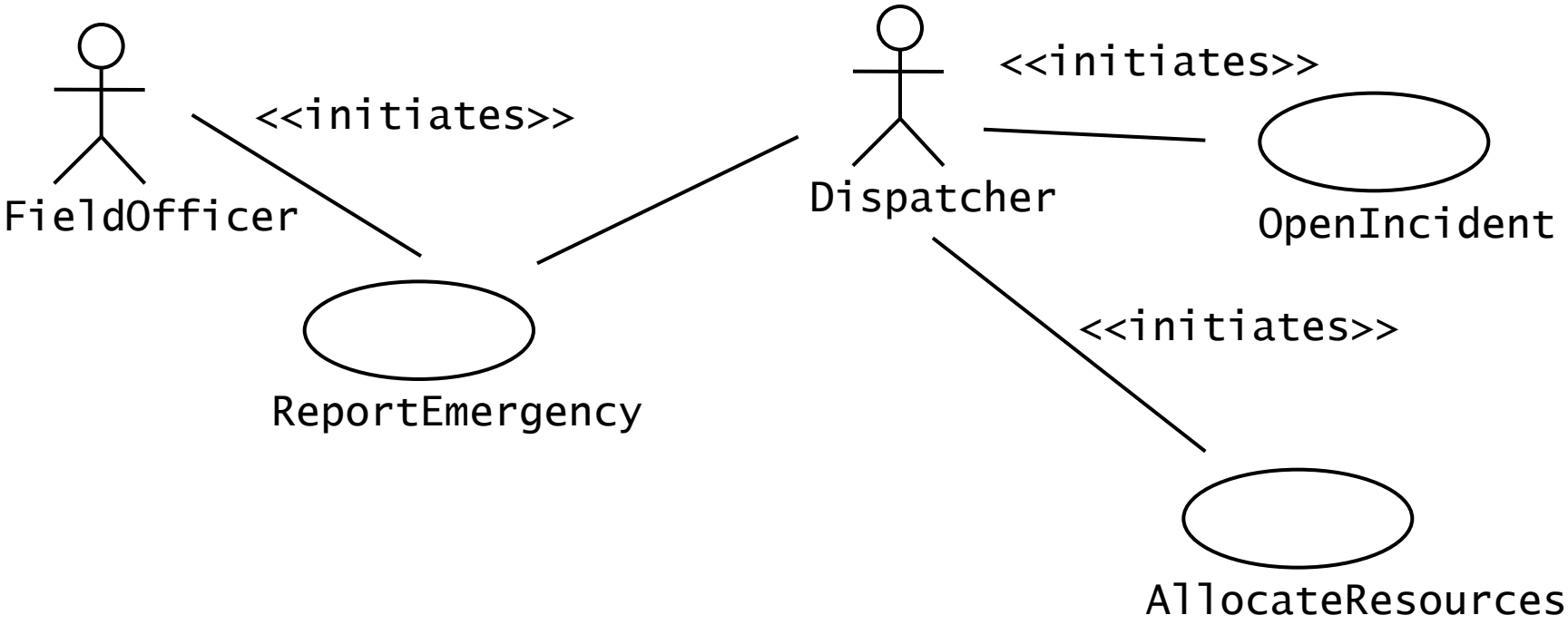
Observations about Warehouse on Fire Scenario

- 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

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.

Another 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

Allocate a Resource (cont'd)

- Use case name: AllocateResources
- Participating Actors:
  Field Officer (Bob and Alice in the Scenario)
  Dispatcher (John in the Scenario)
  Resource Allocator and Field Supervisor
- Entry Condition:
  The Resource Allocator has selected an available resource
- Flow of Events:
  1. The Resource Allocator selects an Emergency Incident
  2. The Resource is committed to the Emergency Incident
- Exit Condition:
  The use case terminates when the resource is committed
  The selected Resource is unavailable to other Requests.
- Special Requirements:
  The Field Supervisor is responsible for managing Resources

Order of steps when formulating use cases

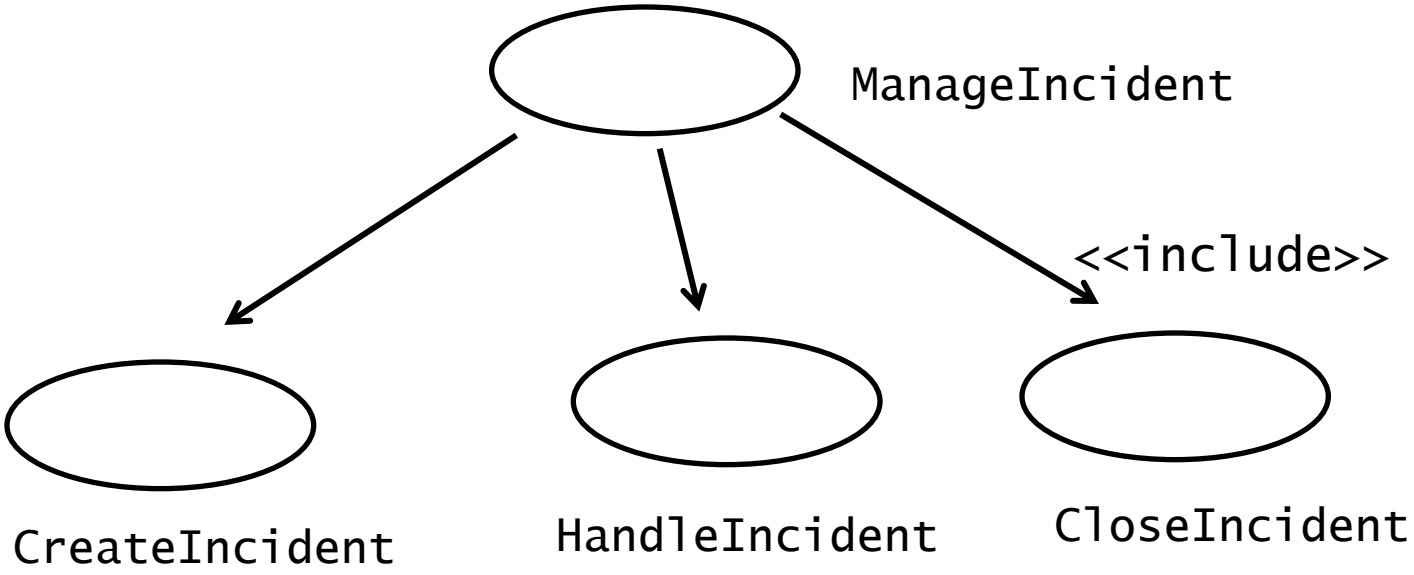
- First step: Name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: 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

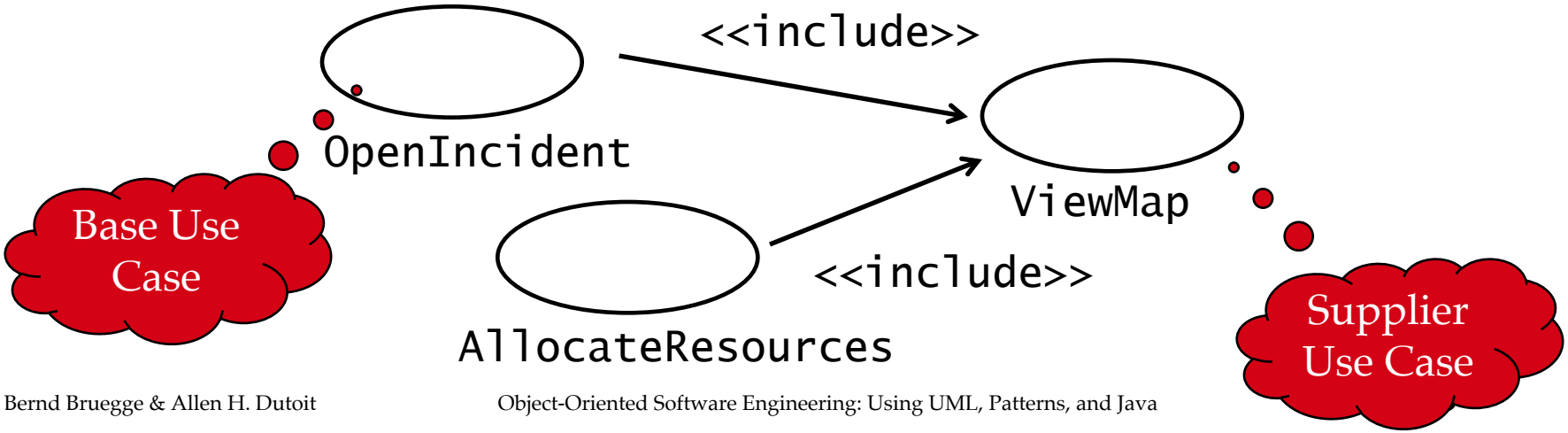
<<include>>: Functional Decomposition

- 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



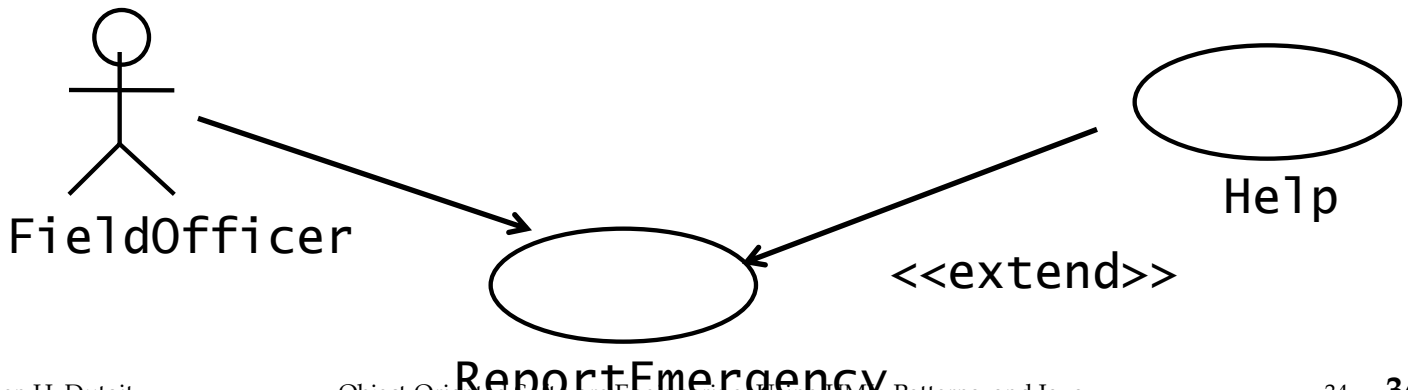
<<include>>: Reuse of Existing Functionality

- 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



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.

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.