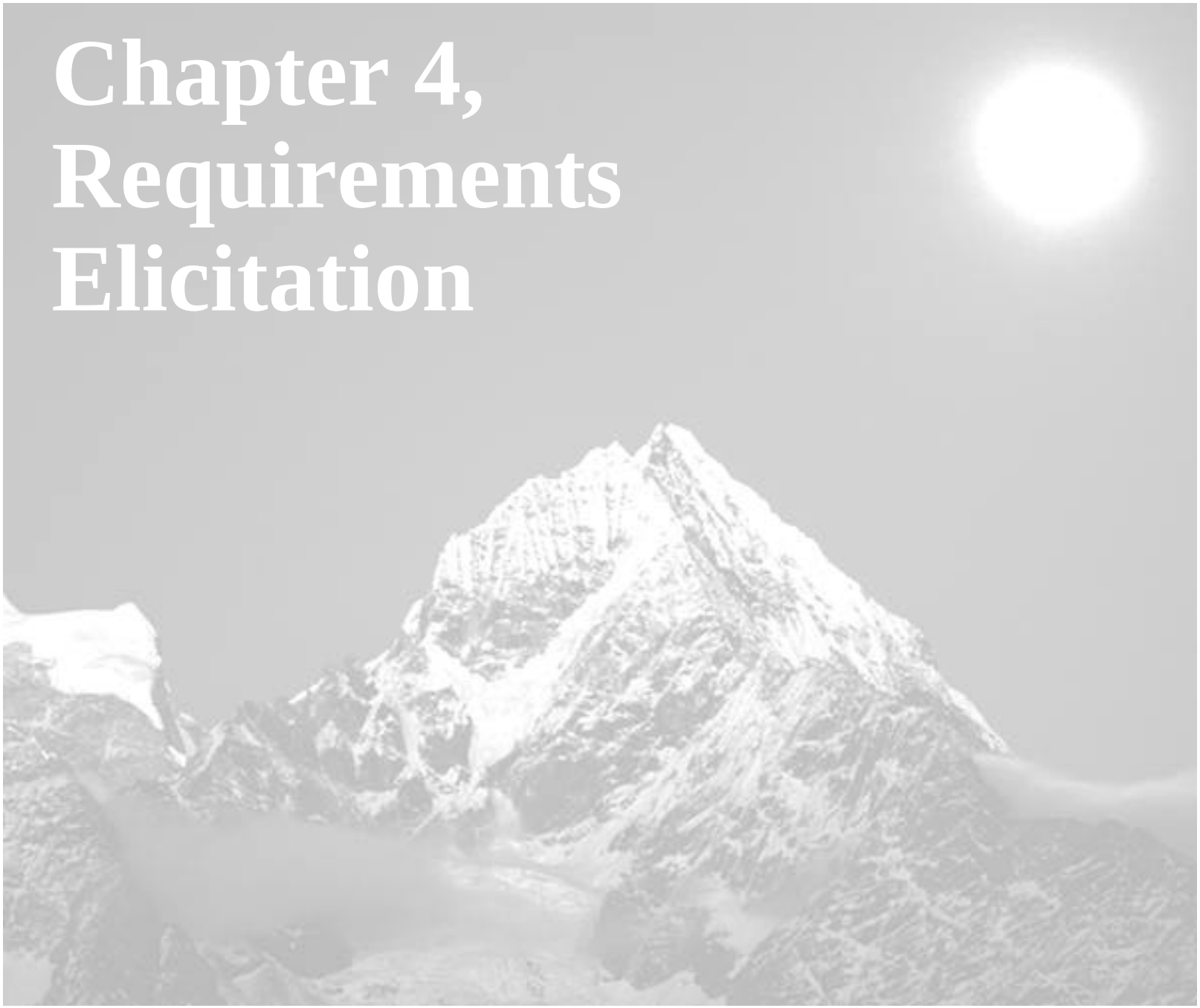
Object-Oriented Software Engineering
Conquering Complex and Changing Systems

Chapter 4, Requirements Elicitation



Preliminaries

- Today:
  - **UML tutorial *(continued)***
  - **Requirements Elicitation lecture**
- Tomorrow: Communication tutorial
  - **Lotus Notes Discussion**
  - **Meeting management**
- Next week
  - **Analysis lecture by Prof. Bruegge**
  - **REQ/QOC tutorial by Allen Dutoit**
- Hauptseminar Requirements Engineering, still available slots
  - **mailto:dutoit@in.tum.de**

Preliminaries (2)

- Office Hours for Helma Schneider *(new!)*
  - **Tuesdays 11:00-12:00**
  - **Fridays 11:00-12:30**
- Account forms
- Magnetic cards for the lab
- Lotus Notes accounts

Defining the System Boundary: What do you see?

Usability Failure

Failure in London Underground:

The driver had taped the button that started the train, relying on the system that prevented the train from moving when the doors were open.

The driver left his train to close a door which was stuck.

When the door finally shut, the train left...

… ***without*** the driver!

Lecture Outline

… some illustrative examples

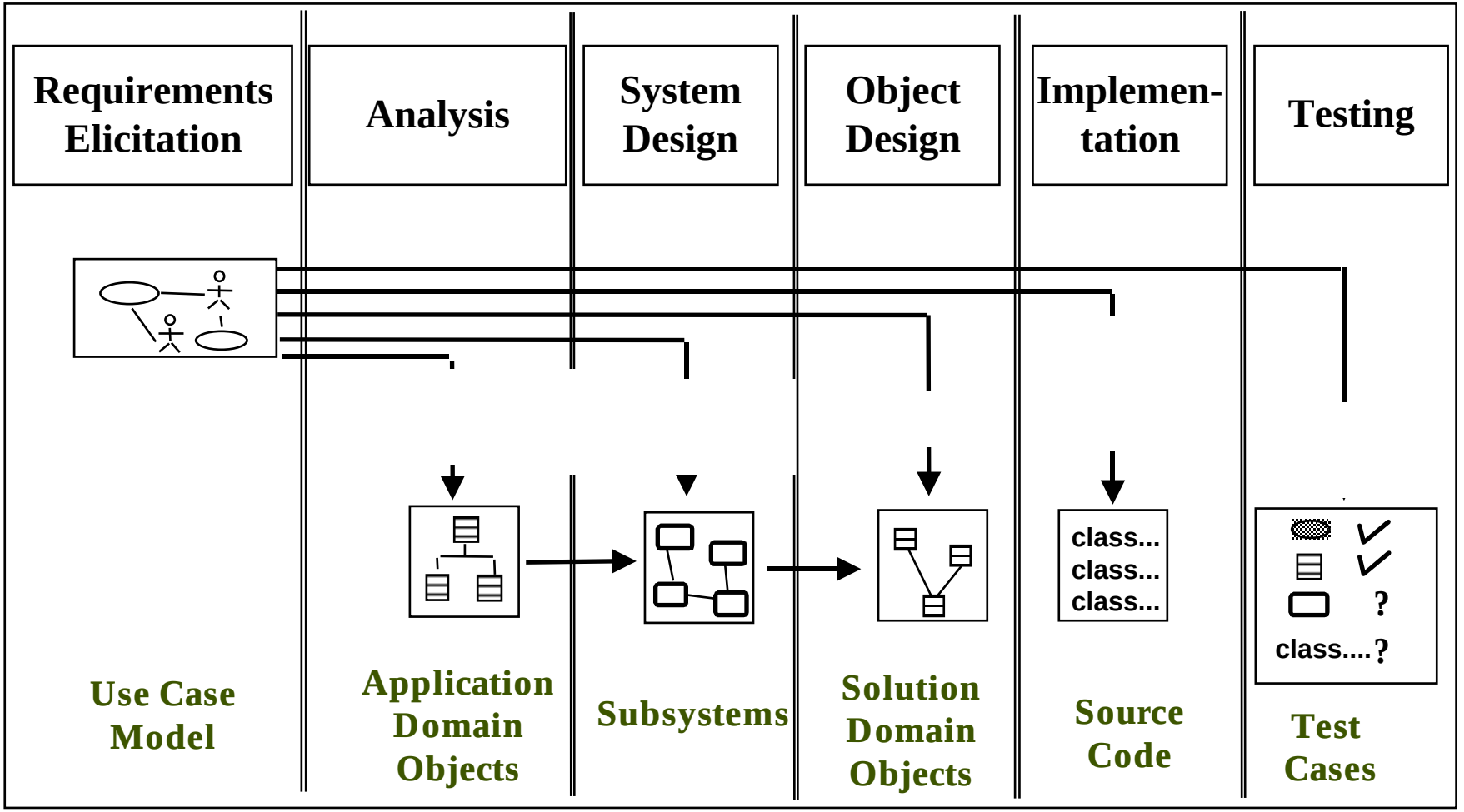
- What is Requirements Elicitation?
- What are Requirements?

… more illustrative examples

- Scenarios
- User Tasks
- Use Cases

- Summary

Software Lifecycle Activities



System Identification

- Development of a system is not simply done by taking a snapshot of a scene (domain)
- Definition of the system boundary
  - **What is inside, what is outside?**
- How can we identify the purpose of a system?

Requirements specification

A requirements specification includes 3 descriptions:

Requirements Elicitation:

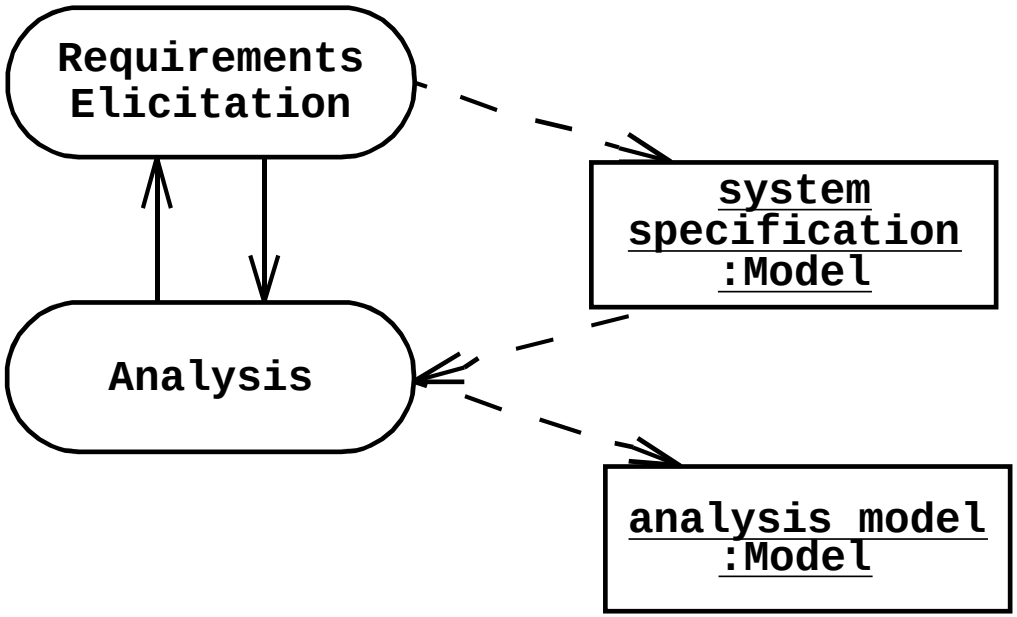
System specification (natural language)

- *Requirements:* What do users do?
- *Interactions:* How do users use the system to accomplish their work?

Analysis:

Requirements analysis model (UML) → *Specification:* What does the system do?

Requirements Process



Requirements Elicitation: Activities

- Identify actors
- Identify scenarios
- Identify use cases
- Identify relationships among use cases
- Refine use cases
- Identify nonfunctional requirements
- Identify participating objects

Requirements Elicitation: Challenges

- Challenging activity
- Requires collaboration of people with different backgrounds
  - **User with application domain knowledge**
  - **Developer with implementation domain knowledge**
- Bridging the gap between user and developer:
  - **Scenarios: Example of the use of the system as a sequence of interactions between the user and the system**
  - **Use cases: Abstraction that describes a class of scenarios**

Types of Requirements

- **Functional requirements**: Interactions between the system and its environment independent from implementation

  The watch system must display the time based on its location

- **Nonfunctional requirements**: User visible aspects of the system not directly related to functional behavior.

  The response time must be less than 1 second

  The accuracy must be within a second

  The watch must be available 24 hours a day except from 2:00am-2:01am and 3:00am-3:01am

- **Constraints** ("Pseudo requirements"): Imposed by the client or the environment in which the system will operate

  The response time must be less than 1 second

  The accuracy must be within a second

  The watch must be available 24 hours a day except from 2:00am-2:01am and 3:00am-3:01am

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

- Critical step in the development process,
  - **Usually after requirements engineering or requirements analysis. Also at delivery**
- Requirements validation criteria:
  - **Correctness:**
    - **The requirements represent the client's view.**
  - **Completeness:**
    - **All possible scenarios through the system are described, including exceptional behavior by the user or the system**
  - **Consistency:**
    - **There are functional or nonfunctional requirements that contradict each other**
  - **Clarity:**
    - **There are no ambiguities in teh requirements.**

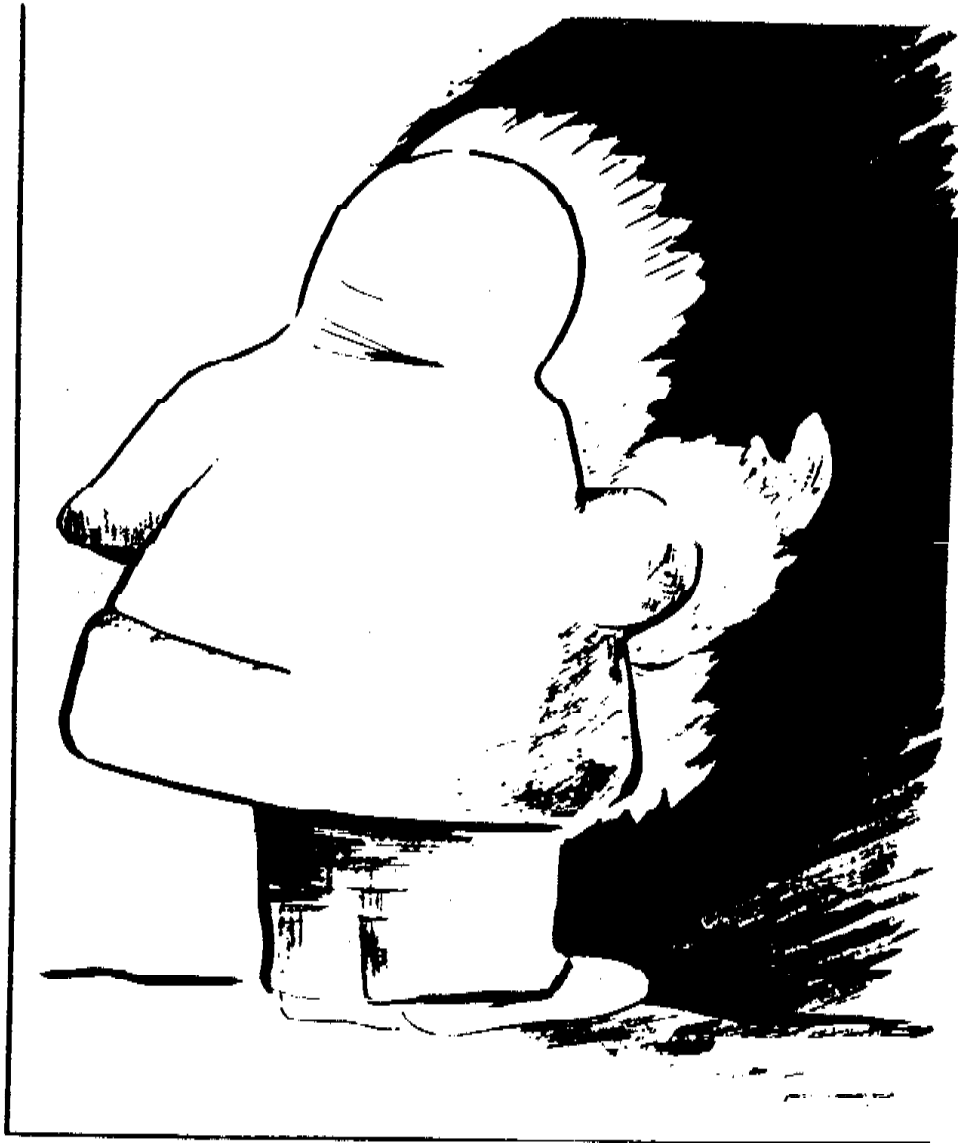
Requirements Validation (continued)

- Realism:
  - **Requirements can be implemented and delivered**
- Traceability:
  - **Each system function can be traced to a corresponding set of functional requirements**

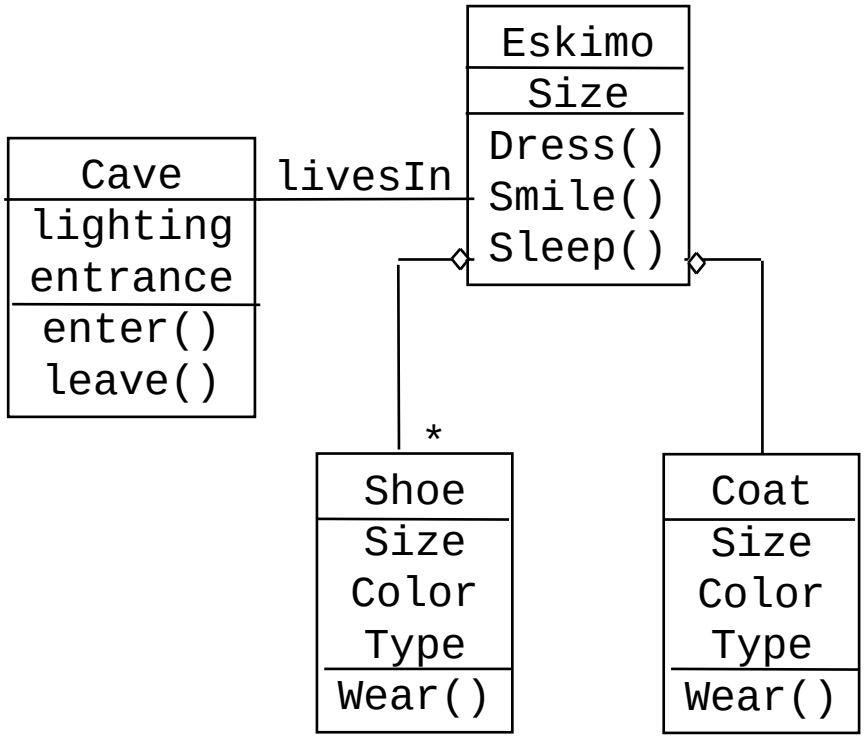
Types of Requirements Elicitation

- 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**
- Re-engineering
  - **Re-design and/or re-implementation of an existing system using newer technology**
  - **Triggered by technology enabler**
- Interface Engineering
  - **Provide the services of an existing system in a new environment**
  - **Triggered by technology enabler or new market needs**

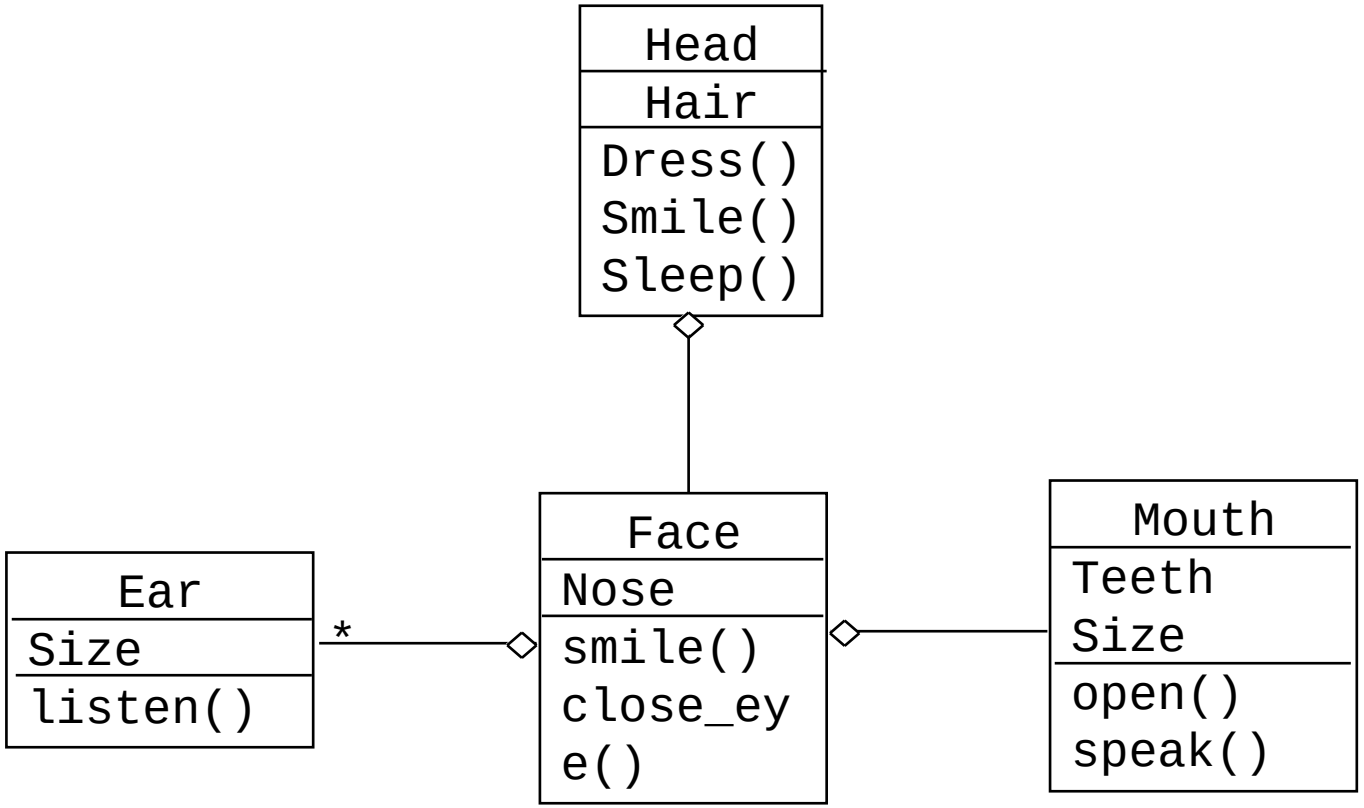
What is This?



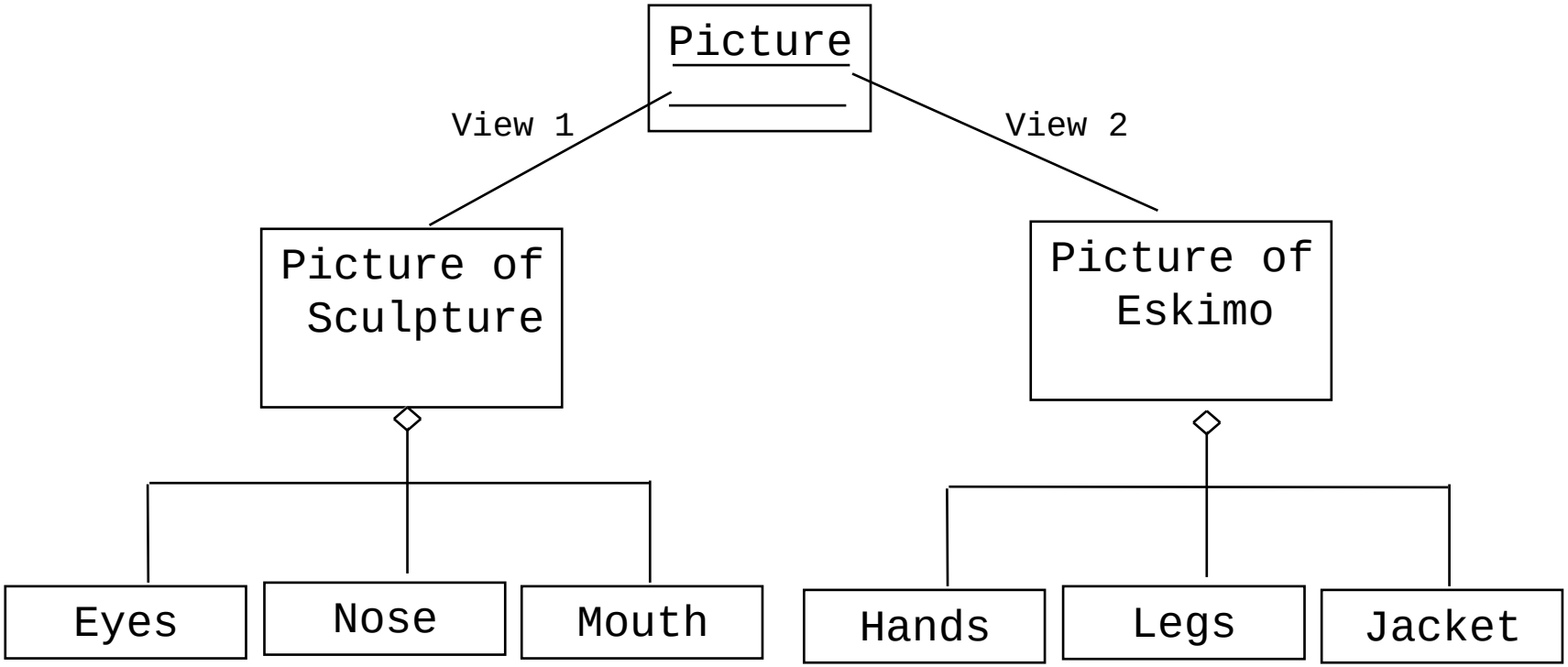
Possible Object Model: Eskimo



Alternative: Head



The Artist's View



Lecture outline

- System Boundary
- What is a Requirements Specification?
- What is Requirements Elicitation?
- Types of Requirements
- *More illustrative examples*
- **Scenarios**
- User Tasks
- Use Cases

System identification

- Two important problems during requirements elicitation and analysis:
  - **Definition of the system purpose**
  - **Identification of objects**
  - **Depending on the purpose of the system, different objects might be found**
    - **What object is inside, what object is outside?**

- How can we identify the purpose of a system?
  - **Scenarios: Examples of system use**
  - **Use cases: Abstractions of scenarios**

Why Scenarios and Use Cases?

- Utterly comprehensible by the user
  - **Use cases model a system from the users' point of view (functional requirements)**
    - **Define every possible event flow through the system**
    - **Description of interaction between objects**
- Great tools to manage a project. Use cases can form basis for whole development process
  - **User manual**
  - **System design and object design**
  - **Implementation**
  - **Test specification**
  - **Client acceptance test**
- An excellent basis for incremental & iterative development
- Use cases have also been proposed for business process reengineering (Ivar Jacobson)

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)
  - **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**

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**

Types of Scenarios

- As-is scenario
  - **Used in describing a current situation. Usually used during re-engineering. The user describes the system.**
- Visionary scenario
  - **Used to describe a future system. Usually described in greenfield engineering or reengineering.**
  - **Can often not be done by the user or developer alone**
- Evaluation scenario
  - **User tasks against which the system is to be evaluated**
- Training scenario
  - **Step by step instructions designed to guide a novice user through a system**

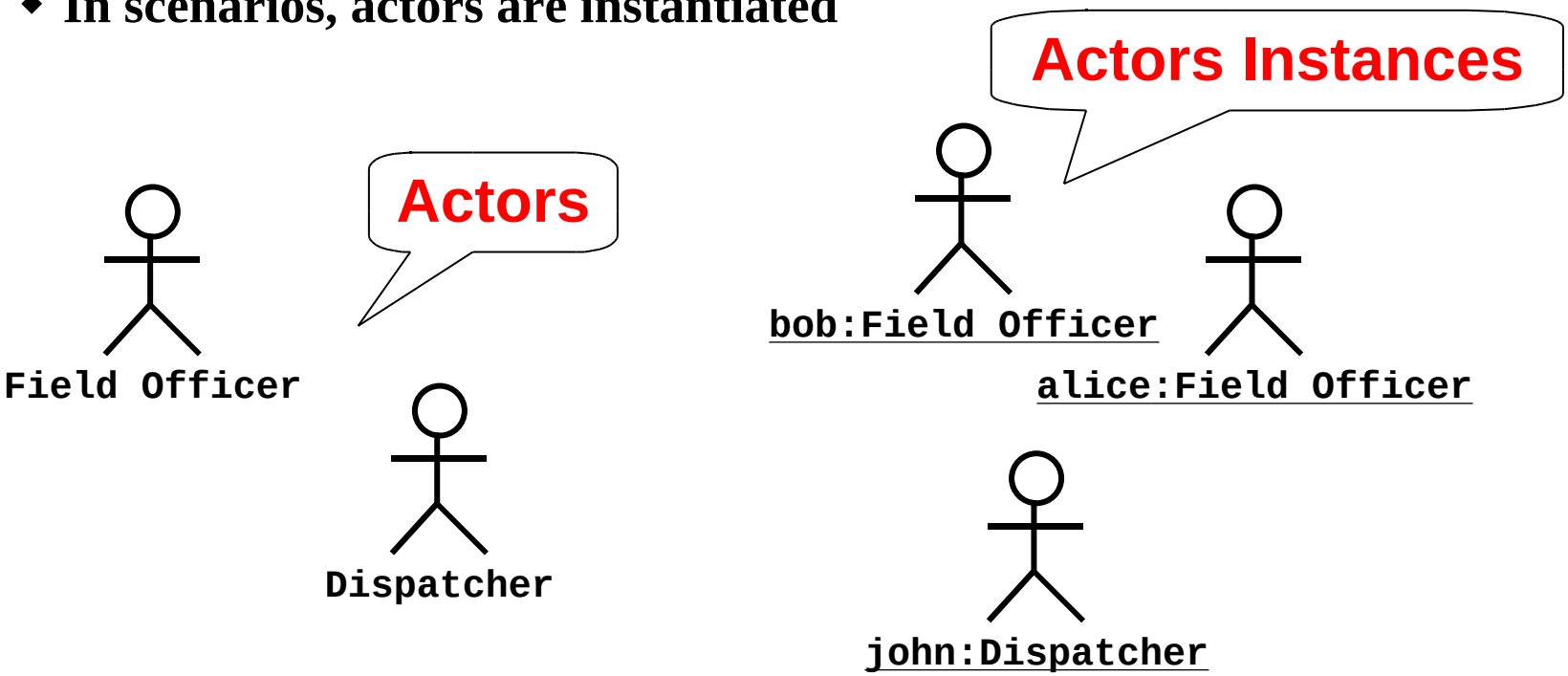
Heuristics for finding Scenarios

- 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?**

- 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**

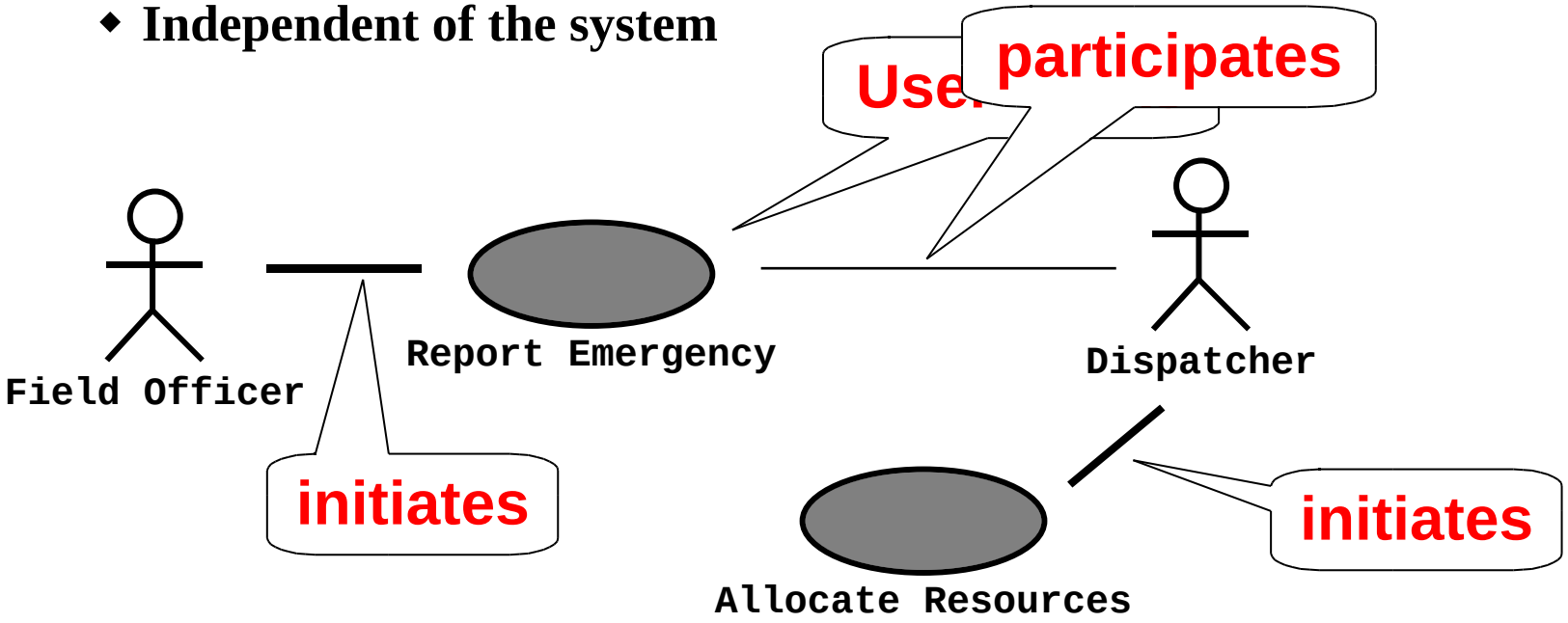
Outline Requirements with Actors and User Tasks

- Actors (see UML Lecture):
  - **Represent an entity outside the system**
  - **Roles are represented as different actors**
  - **External systems are also represented as actors**
  - **In scenarios, actors are instantiated**



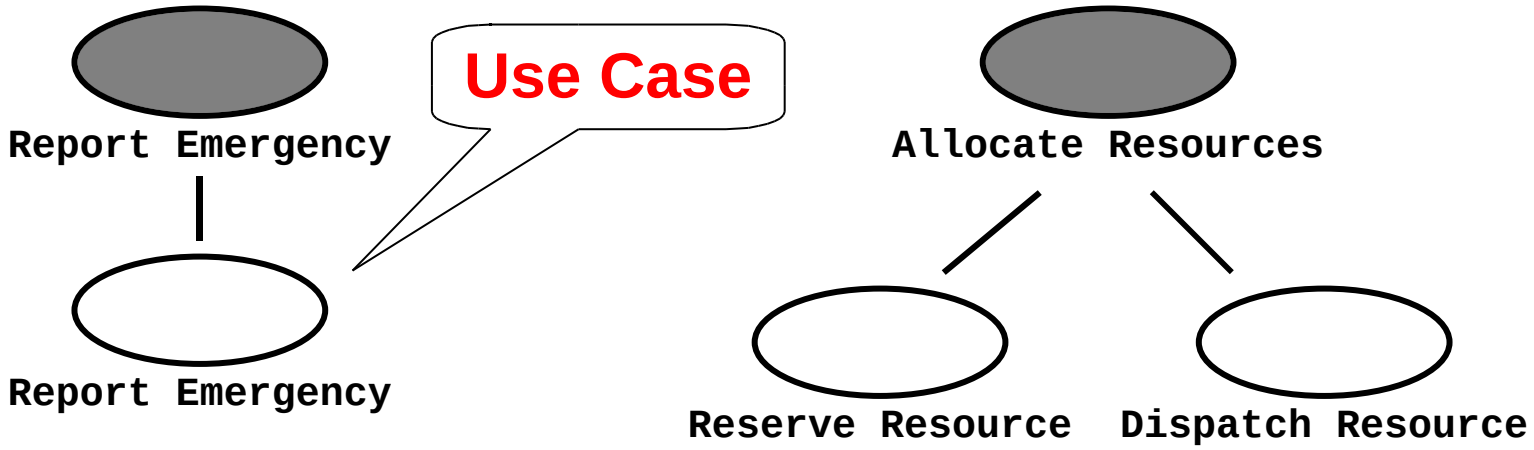
Outline Requirements with User Tasks (2)

- User Tasks:
  - **High-level descriptions of the user's work**
  - **Initiated by an actor *(initiator)***
  - **May involve other actors *(participating actors)***
  - **Independent of the system**



Describe Interactions with Use Cases

- Use Cases (see UML Lecture)
  - **Detailed description of interactions between users and system**
  - **Specifies all possible scenarios**
  - **Will be used to identify objects during analysis**
  - **Realizes part or all of a User Task**



Describe Interactions with Use Cases (2)

- Find a use case 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 this use case in more detail
  - **Describe the flow of events**
  - **Describe the entry condition**
  - **Describe the exit condition**
  - **Describe exceptions**
  - **Describe special requirements (constraints, nonfunctional requirements)**

Example of steps in formulating a use case

Formulate the flow of events:

- The FieldOfficer activates the “Report Emergency” function on 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.

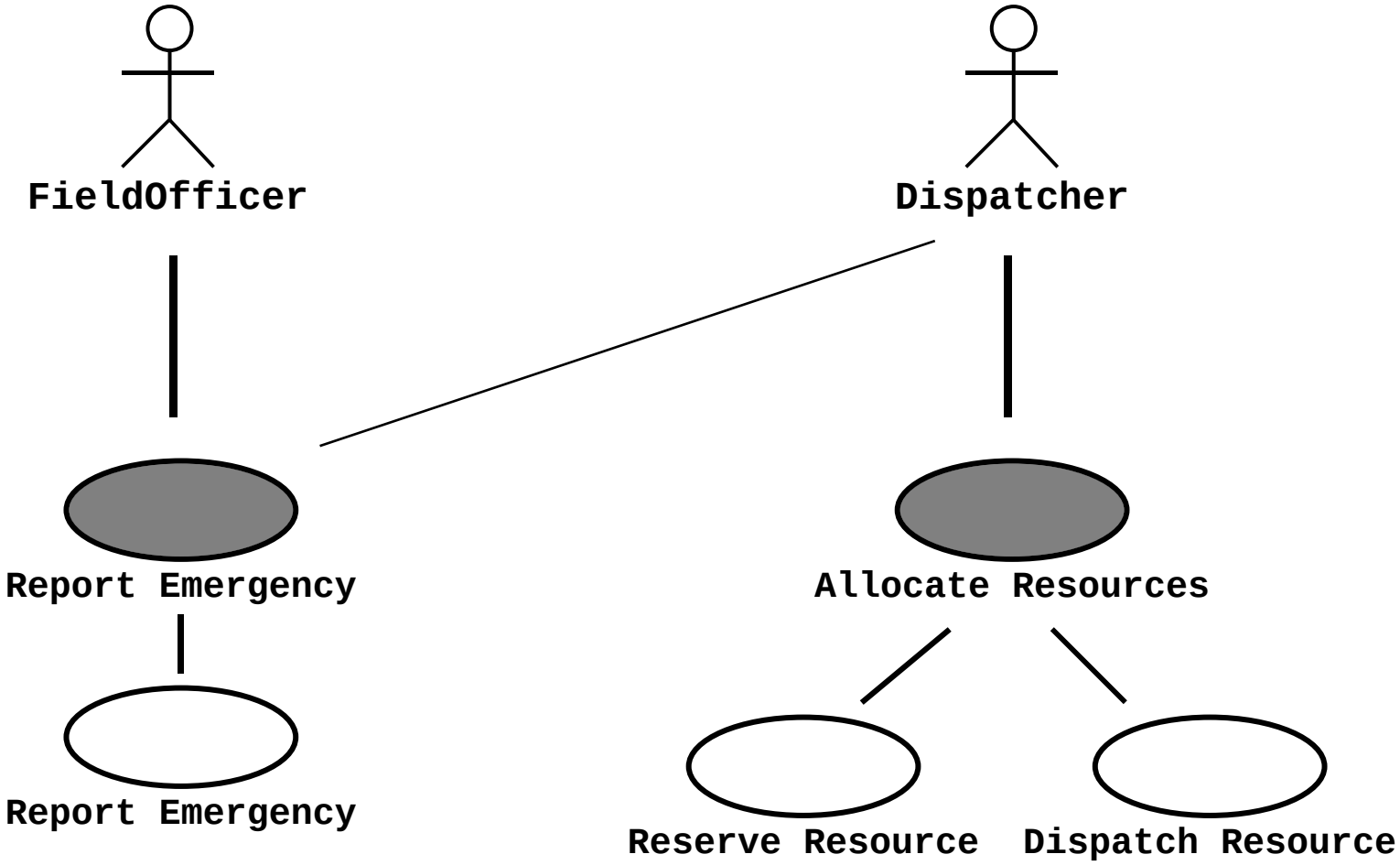
Example of steps in formulating a use case

- Write down the 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.**
- Identify and write down any 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.**
- Associate the Use Case with the corresponding User Task
  - **Report Emergency Use Case realizes the Report Emergency User Task**

How to Specify a Use Case (Summary)

- Name of Use Case
- Realized User Task
  - **Reference to user task that this use case realizes.**
- Entry condition
  - **Use a syntactic phrase such as “This use case starts when…”**
- Flow of Events
  - **Free form, informal natural language**
- Exit condition
  - **Star with “This use cases terminates when…”**
- Exceptions
  - **Describe what happens if things go wrong**
- Special Requirements
  - **List nonfunctional requirements and constraints**

Use Case Model for Incident Management

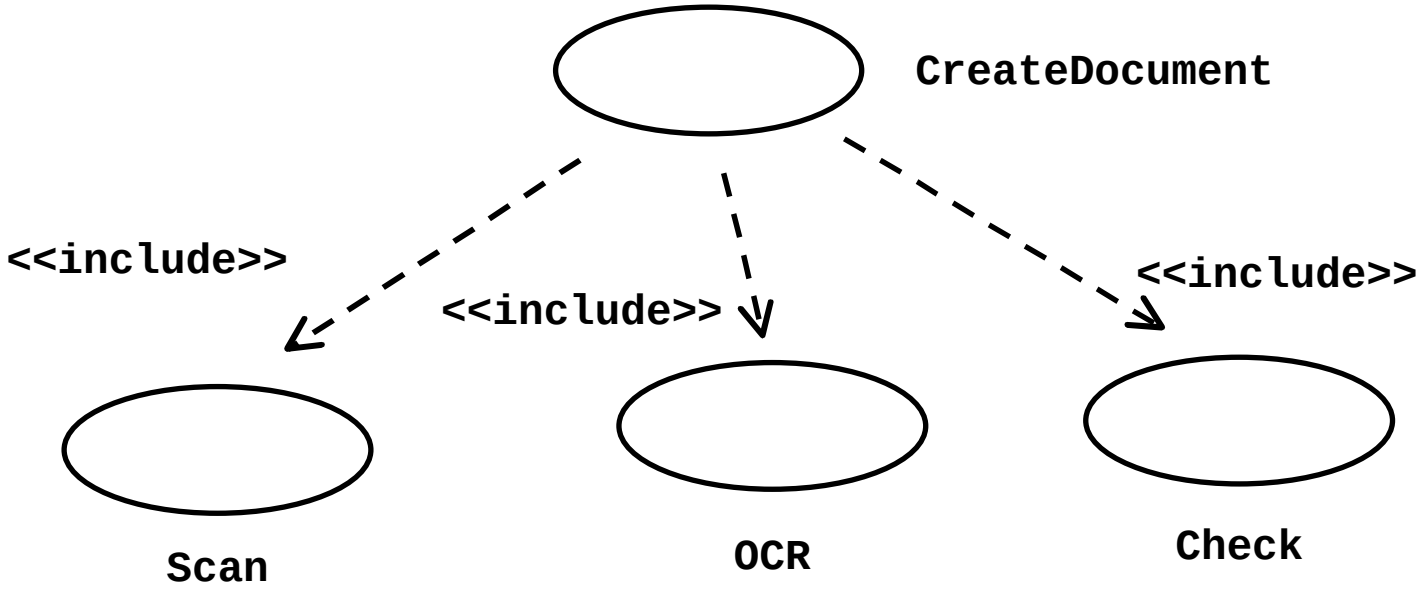


Use Case Associations

- Use case association = relationship between use cases
- Important types:
  - **Extends**
    - **A use case extends another use case**
  - **Include**
    - **A use case uses another use case (“functional decomposition”)**
  - **Generalization**
    - **An abstract use case has different specializations**

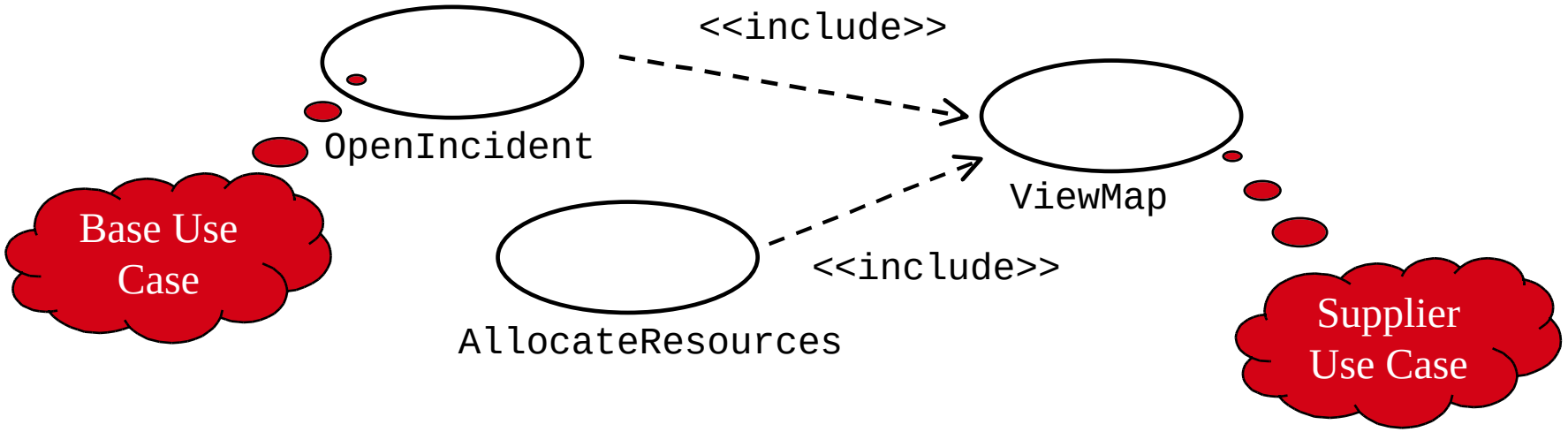
<<Include>>: Functional Decomposition

- 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**



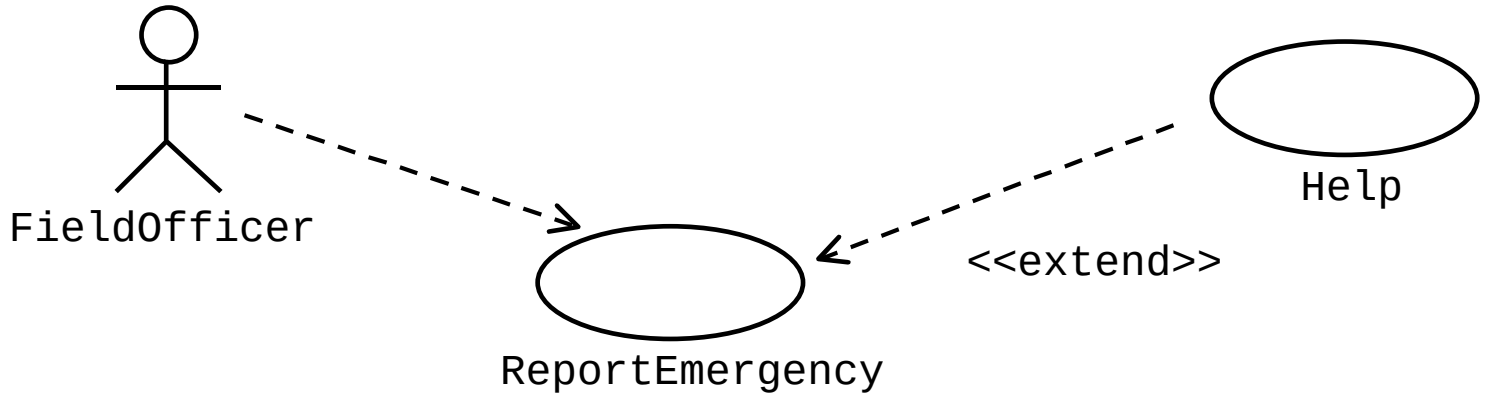
<<Include>>: Reuse of Existing Functionality

- 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**



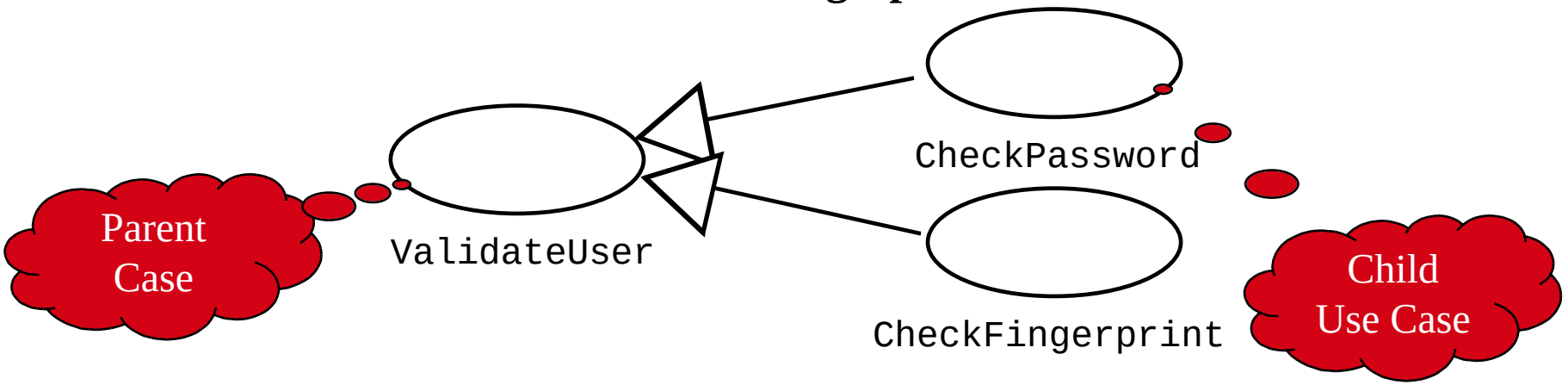
<<Extend>> Association for Use Cases

- 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: In an extend assocation, the base use case can be executed without the use case extension



Generalization association in use cases

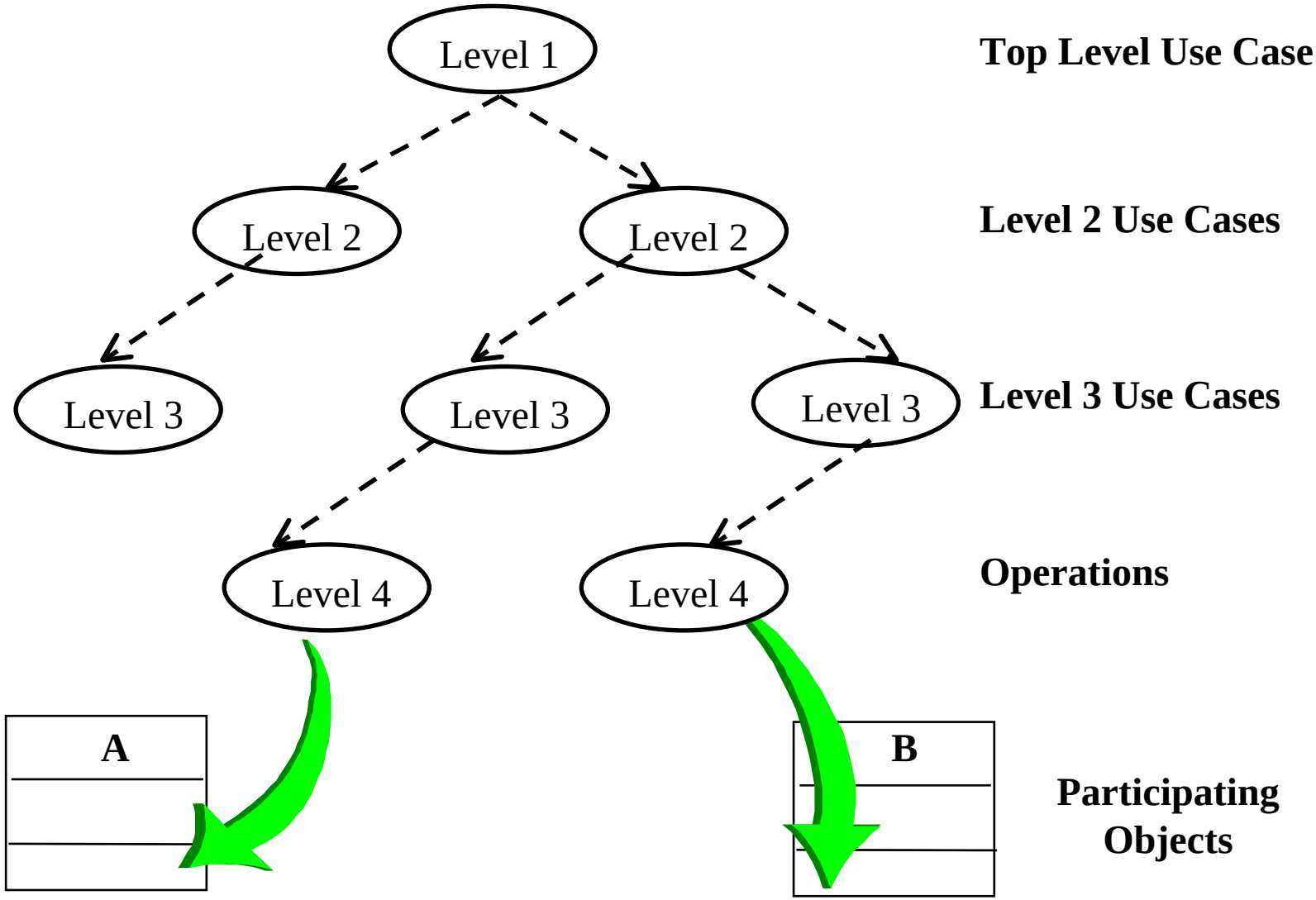
- Problem:
  - **You have common behavior among use cases and want to factor this out.**
- Solution:
  - **The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.**
- Example: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



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 mock-ups as visual support
- Find out what the user does
  - **Questionnaires (Good)**
  - **Task observation (Better)**

From Use Cases to Objects



Finding Participating Objects in Use Cases

- For any use case do the following
  - **Find terms that developers or users need to clarify in order to understand the flow of events**
    - **Always start with the user's terms, then negotiate:**
      - **FieldOfficerStationBoundary or FieldOfficerStation?**
      - **IncidentBoundary or IncidentForm?**
      - **EOPControl or EOP?**
  - **Identify real world entities that the system needs to keep track of. Examples: FieldOfficer, Dispatcher, Resource**
  - **Identify real world procedures that the system needs to keep track of. Example: EmergencyOperationsPlan**
  - **Identify data sources or sinks. Example: Printer**
  - **Identify interface artifacts. Example: PoliceStation**
  - **Do textual analysis to find additional objects (Use Abott's technique)**
  - **Model the flow of events with a sequence diagram**

Summary

- Requirements Elicitation:
  - **Greenfield Engineering, Reengineering, Interface Engineering**
- Scenarios:
  - **Supports communication with client**
  - **As-Is scenarios, Visionary scenarios, Evaluation scenarios**
  - **User tasks: high level abstractions of scenarios**
  - **Use cases: detailed abstractions of scenarios**
- Pure functional decomposition is bad:
- Pure object identification is bad:
  - **May lead to wrong objects, wrong attributes, wrong methods**
- The key to successful analysis:
  - **Start with scenarios, user tasks, and use cases**
  - **Then find the participating objects**
  - **If somebody asks “What is this?”, do not answer right away. Ask (or observe): “What is it used for?”**

Exercises (solutions next Thursday)

2.6 Draw a sequence diagram for the `warehouseOnFire` scenario (as described in this presentation). Include the objects bob, alice, john, system, and instances of other classes you may need. Draw only the first five message sends.

2.7 Draw a sequence diagram for the `ReportIncident` use case (as described in this lecture). Make sure it is consistent with the sequence diagram of the previous exercise. Draw only the first five message sends.