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

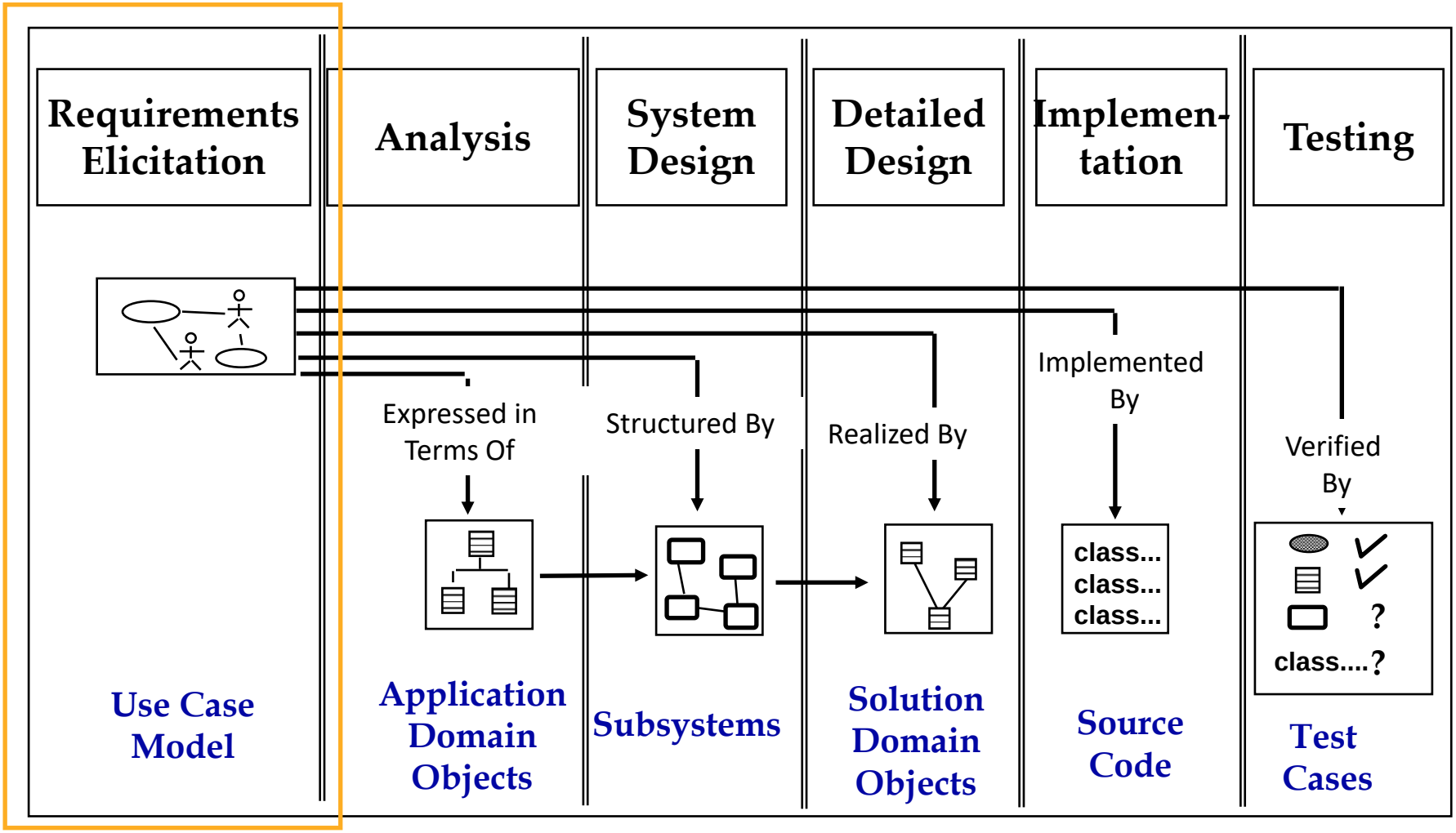
Chapter 4, Requirements Elicitation

Compiled by Dr. Osama Hosameldeen

Contents

- Definition of Requirement Eliciation
- Problem statement
- Scenario-based Design
- Types of requirements
- Requirements Validation
- Scenario example from earlier: Warehouse on Fire

Software Lifecycle Activities



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 interface
4. Glossary

What does the Customer say?

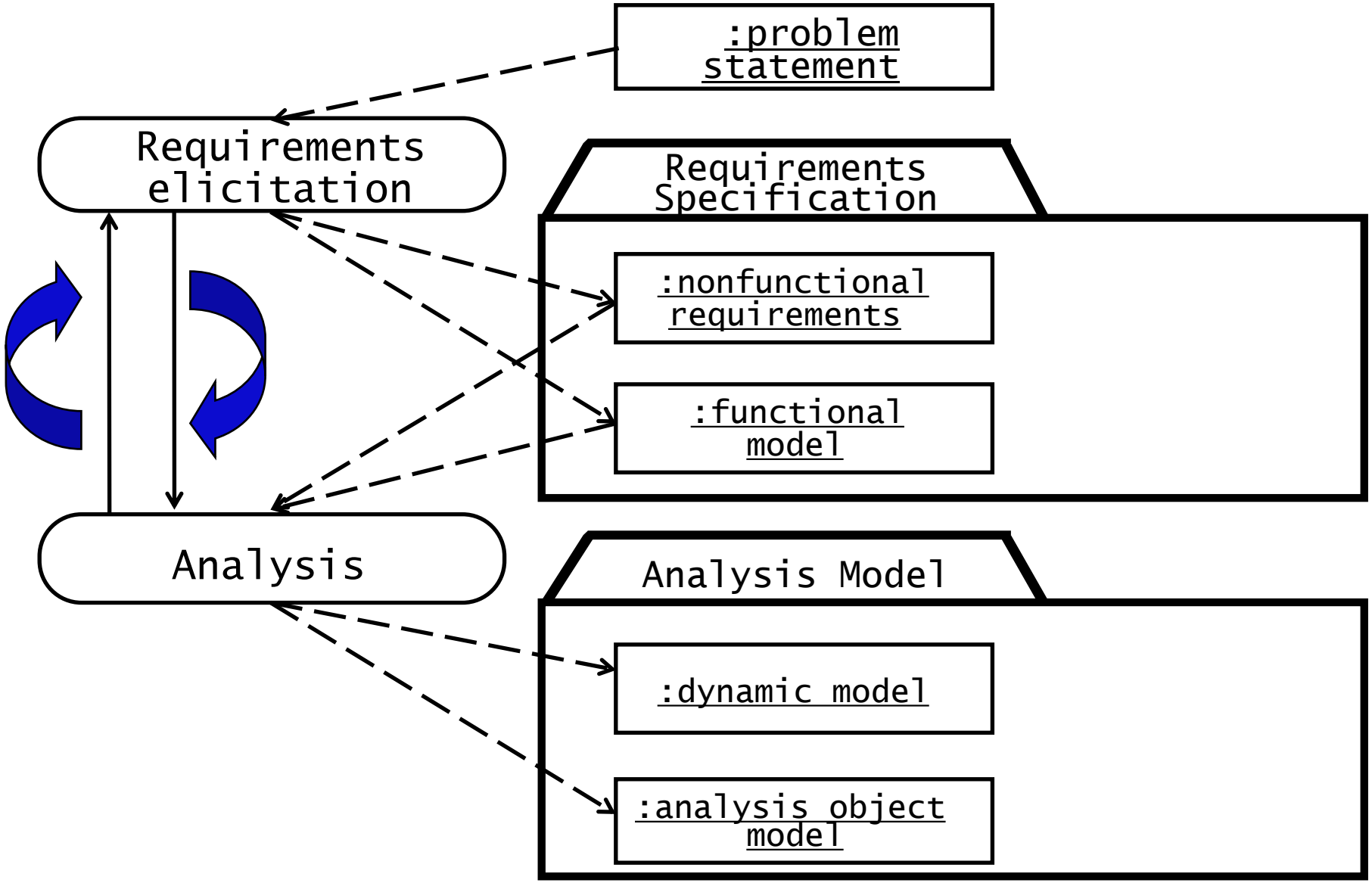
First step in identifying the Requirements: System identification

- 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.

Different Types of Requirements Elicitation

- Greenfield Engineering
  - Development starts from scratch, no prior system exists, requirements come from end users and clients
  - 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
  - Provision of existing services in a new environment
  - Triggered by technology enabler or new market needs

Requirements Process



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.

Contents

- Definition of Requirement Eliciation
- **Problem statement**
- Scenario-based Design
- Types of requirements
- Requirements Validation
- Scenario example from earlier: Warehouse on Fire

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 chess is far too slow.
    - I want to play chess, 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.

Contents

- Definition of Requirement Eliciation
- Problem statement
- **Scenario-based Design**
- Types of requirements
- Requirements Validation
- Scenario example from earlier: Warehouse on Fire

Scenario-Based Design

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

Scenario-Based Design: The use of scenarios in a software lifecycle activity

Types of Scenarios

- As-is scenario:
  - Describes a current situation. Usually used in re-engineering projects. The user describes the system
    - Example: Description of 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.

Additional Types of Scenarios (2)

- Evaluation scenario:
  - Description of a user task 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:
  - A description of the step by step instructions that guide a novice user through a system
    - Example: How to play Tic Tac Toe in the ARENA Game Framework.

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.

Contents

- Definition of Requirement Eliciation
- Problem statement
- Scenario-based Design
- **Types of requirements**
- Requirements Validation
- Scenario example from earlier: Warehouse on Fire

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.

Types of Nonfunctional Requirements

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."
  - *Usability Requirement*
- "The system must be running 95% of the time"
  - *Reliability Requirement*
- "The system must support 10 parallel tournaments"
  - *Performance Requirement*
- "The operator must be able to add new games without modifications to the existing system."

Quality requirements

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

Constraints or Pseudo requirements

Contents

- Definition of Requirement Eliciation
- Problem statement
- Scenario-based Design
- Types of requirements
- **Requirements Validation**
- Scenario example from earlier: Warehouse on Fire

Requirements Validation

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis

- Correctness:
  - The requirements represent the client's view
- Completeness:
  - All possible scenarios, in which the system can be used, are described
- Consistency:
  - There are no requirements that contradict each other.

Requirements Validation (2)

- Clarity:
  - Requirements can only be interpreted in one way
- Realism:
  - Requirements can be implemented and delivered
- Traceability:
  - Each system behavior can be traced to a set of functional requirements

- Problems with requirements validation:
  - Requirements change quickly during requirements elicitation
  - Inconsistencies are easily added with each change
  - Tool support is needed!

Contents

- Definition of Requirement Eliciation
- Problem statement
- Scenario-based Design
- Types of requirements
- Requirements Validation
- **Scenario example from earlier: Warehouse on Fire**

Scenario example from earlier: 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

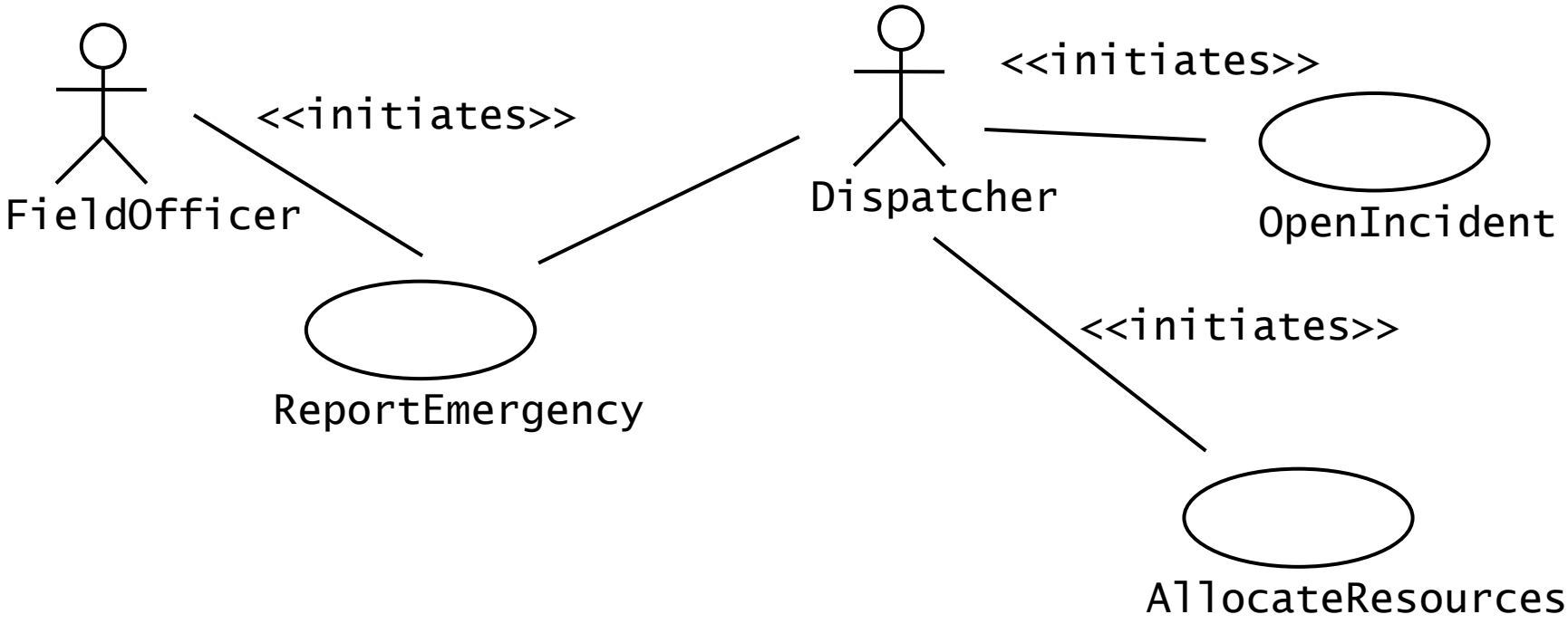
- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and John

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

Scenario example

Use Case Model for Incident Management



Use Case Example: ReportEmergency Flow of Events

1. The **FieldOfficer** activates the "Report Emergency" function of her terminal. The system 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

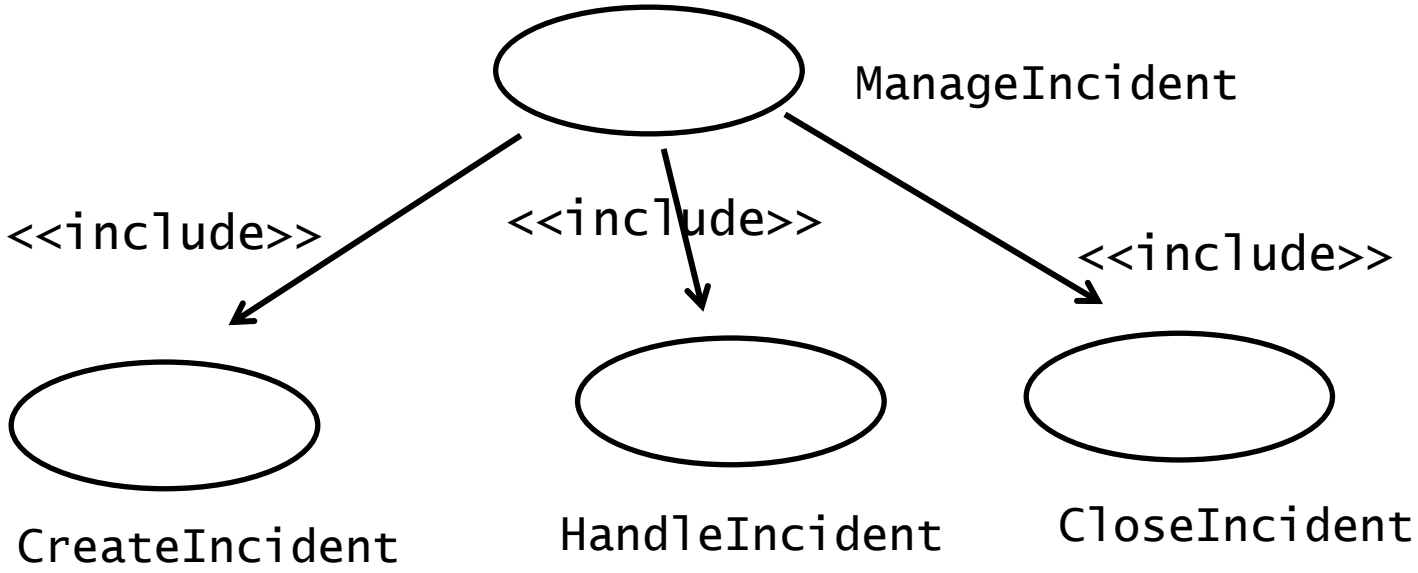
Use Case Associations

- 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

Scenario example

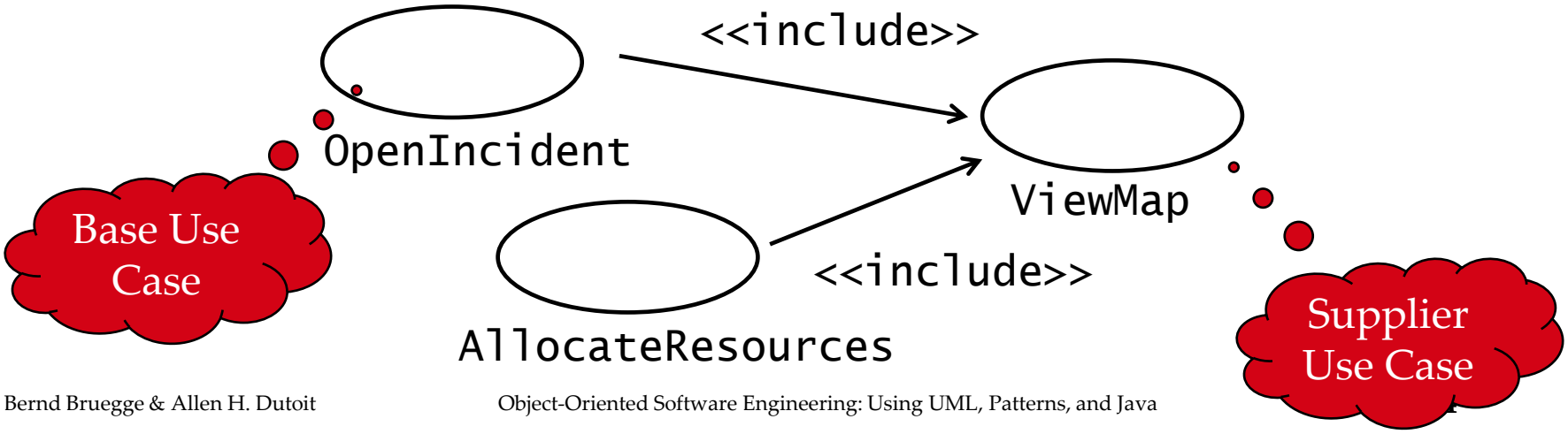
<<include>>: Functional Decomposition

- 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



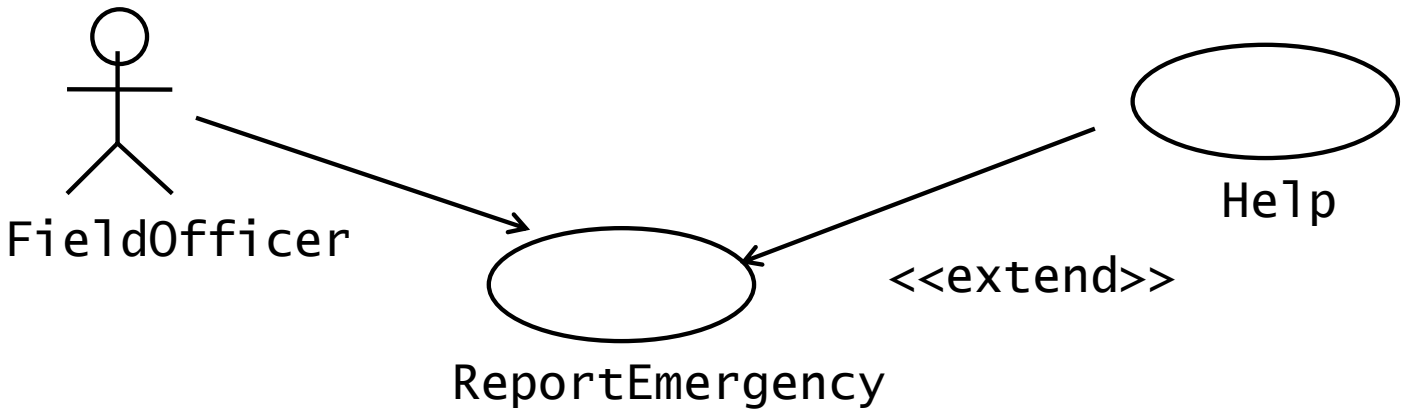
<<include>>: Reuse of Existing Functionality

- 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)



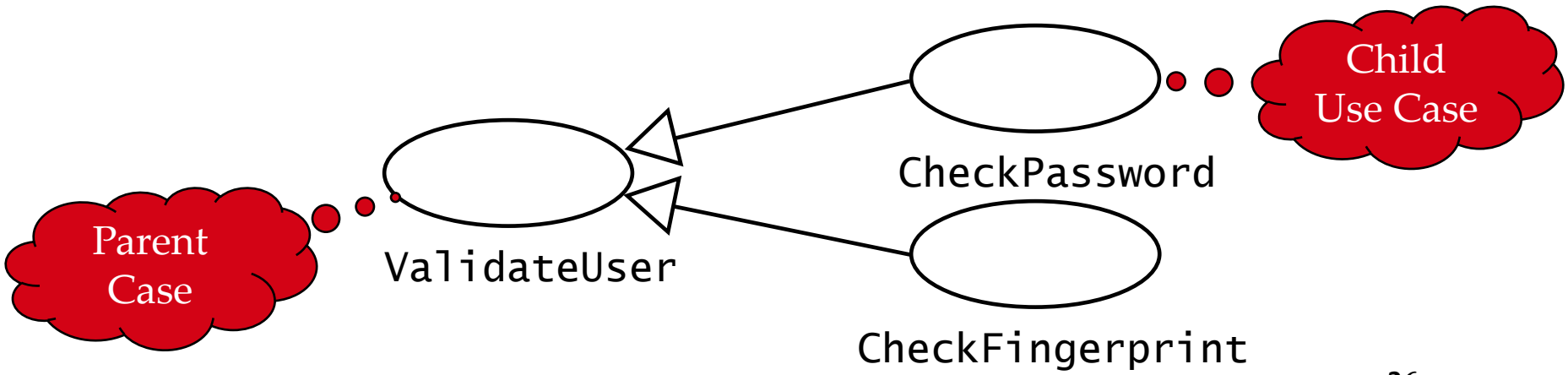
<<extend>> Association for Use Cases

- 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 may 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"



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.