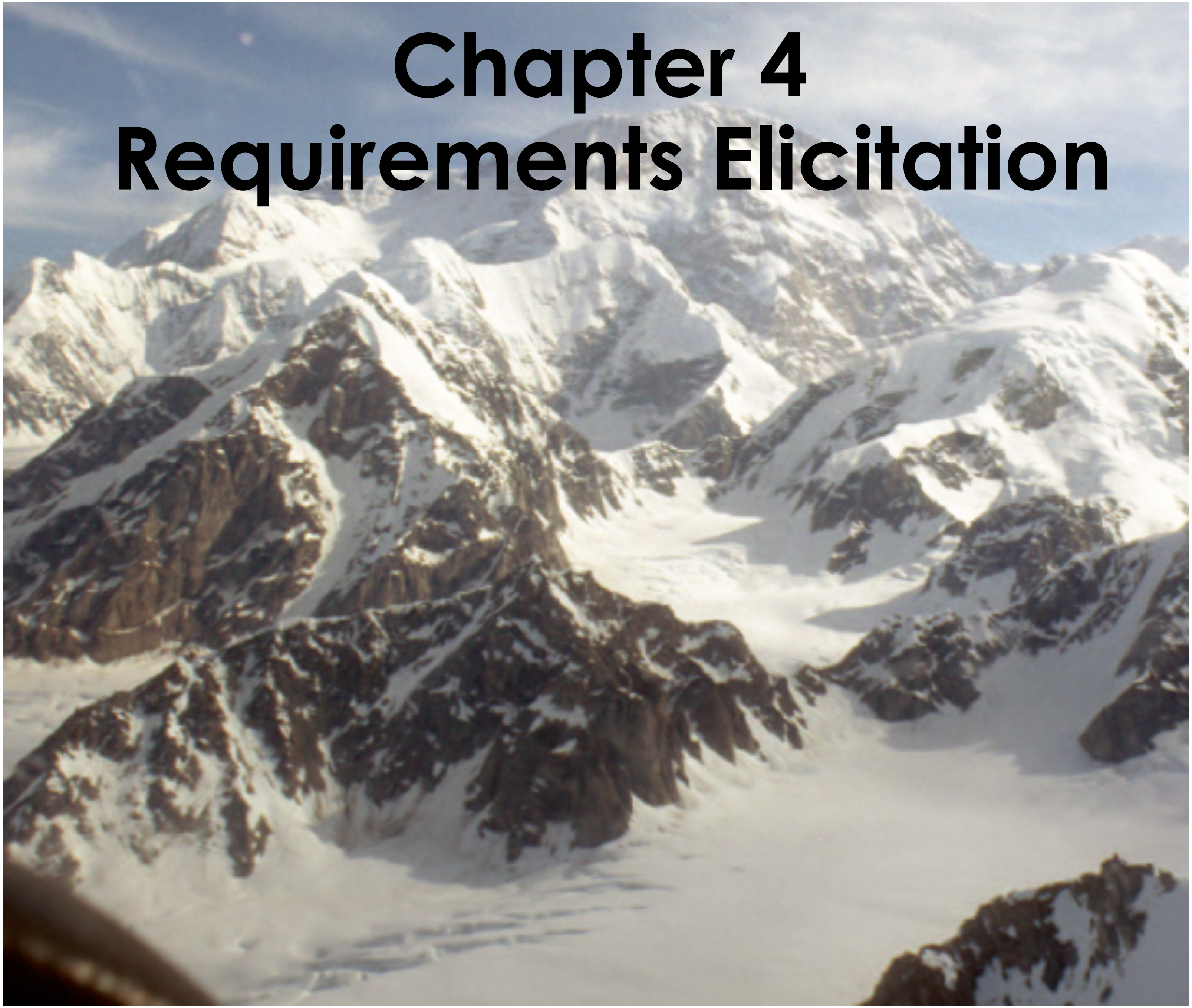
Object-Oriented Software Engineering
Using UML, Patterns, and Java



Chapter 4
Requirements Elicitation

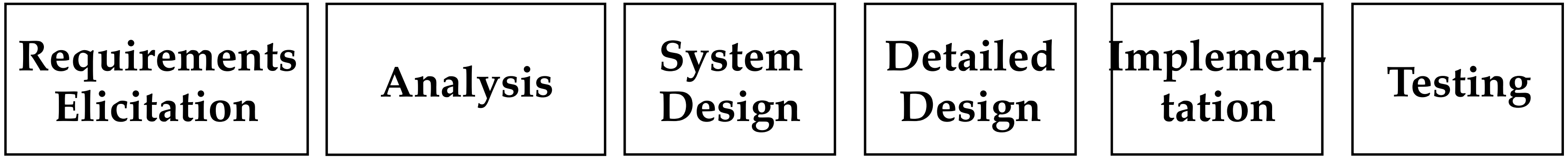
Outline

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

Software Lifecycle Definition

- Software lifecycle
  - Models for the development of software
    - Set of **activities and** their **dependency relationship**s to each other to support the development of a software system
    - Examples:
      - Analysis, design, implementation, testing
      - Design depends on analysis, testing can be done before implementation

A Typical Example of Software Lifecycle Activities



Requirements Elicitation

Analysis

System Design

Detailed Design

Implemen-tation

Testing

Software Lifecycle Activities...and their models

Requirements Elicitation

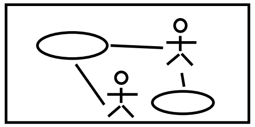
Analysis

System Design

Detailed Design

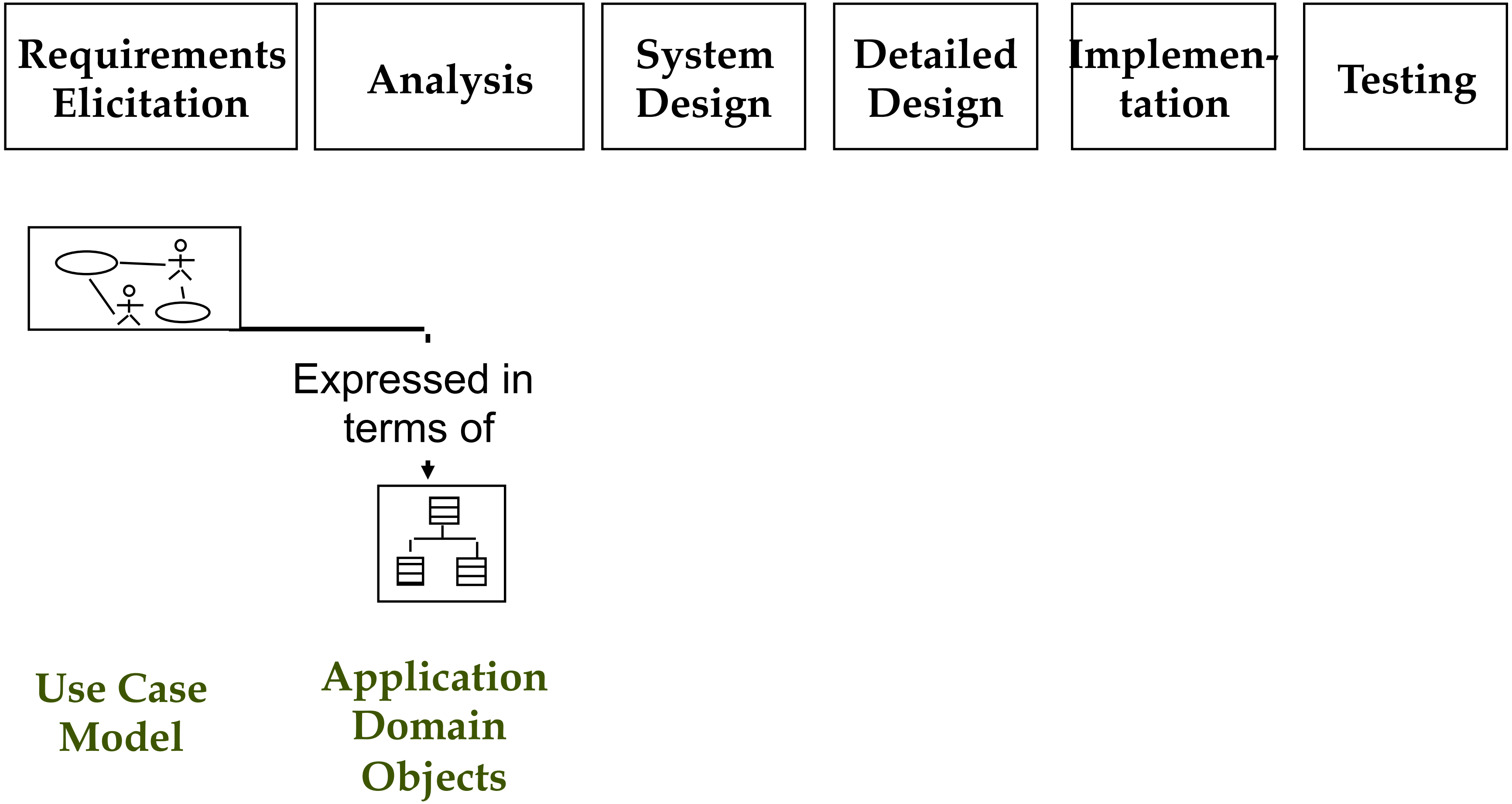
Implementation

Testing



Use Case Model

Software Lifecycle Activities...and their models



Requirements Elicitation

Analysis

System Design

Detailed Design

Implementation

Testing

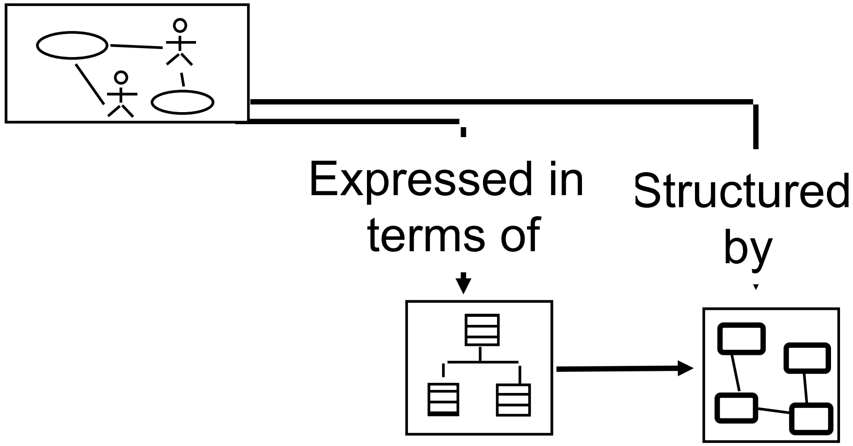
Expressed in terms of

Use Case Model

Application Domain Objects

Software Lifecycle Activities…and their models

Requirements Elicitation | Analysis | System Design | Detailed Design | Implementation | Testing

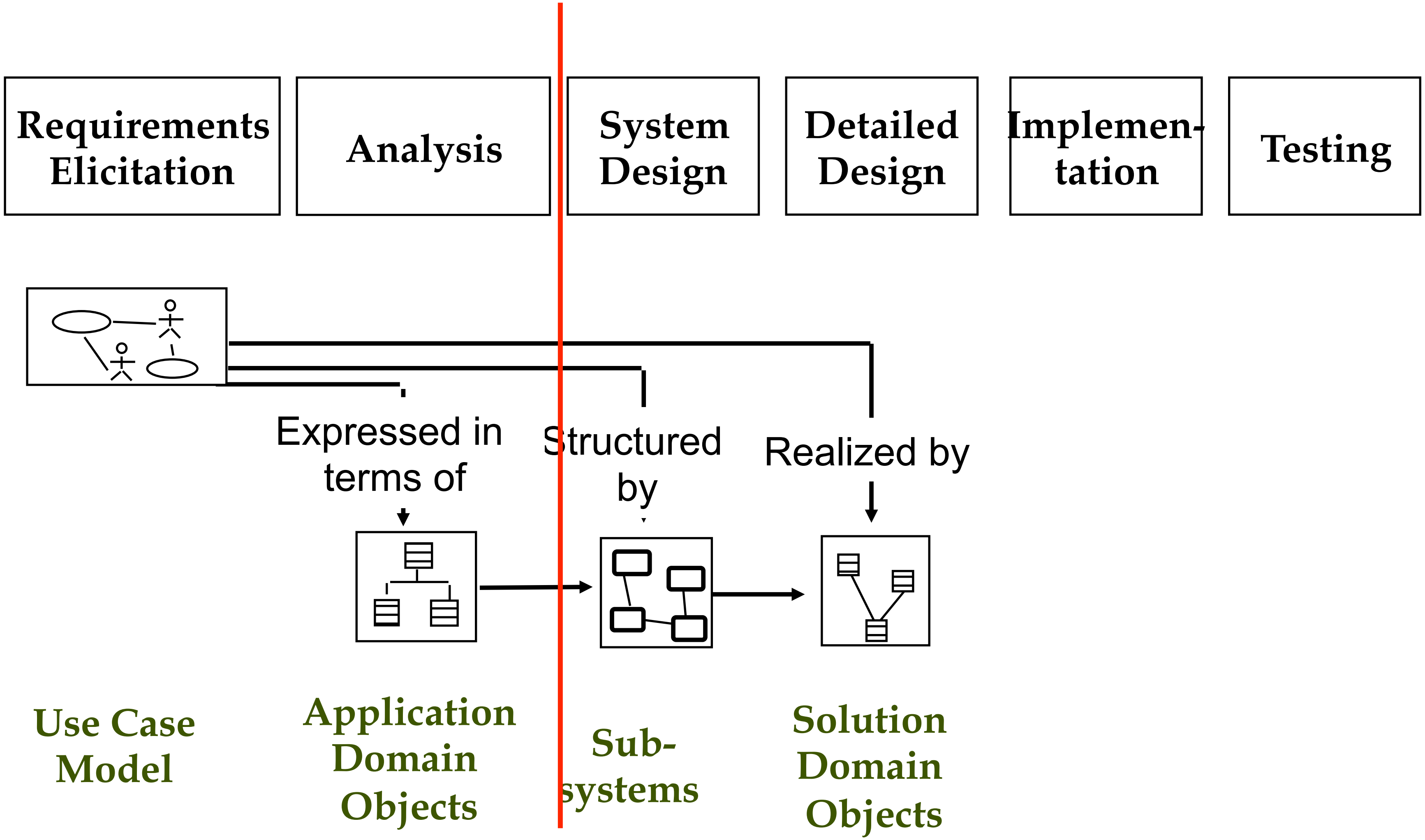


Use Case Model

Application Domain Objects

Sub-systems

Software Lifecycle Activities...and their models



Requirements Elicitation

Analysis

System Design

Detailed Design

Implemen-tation

Testing

Expressed in terms of

Structured by

Realized by

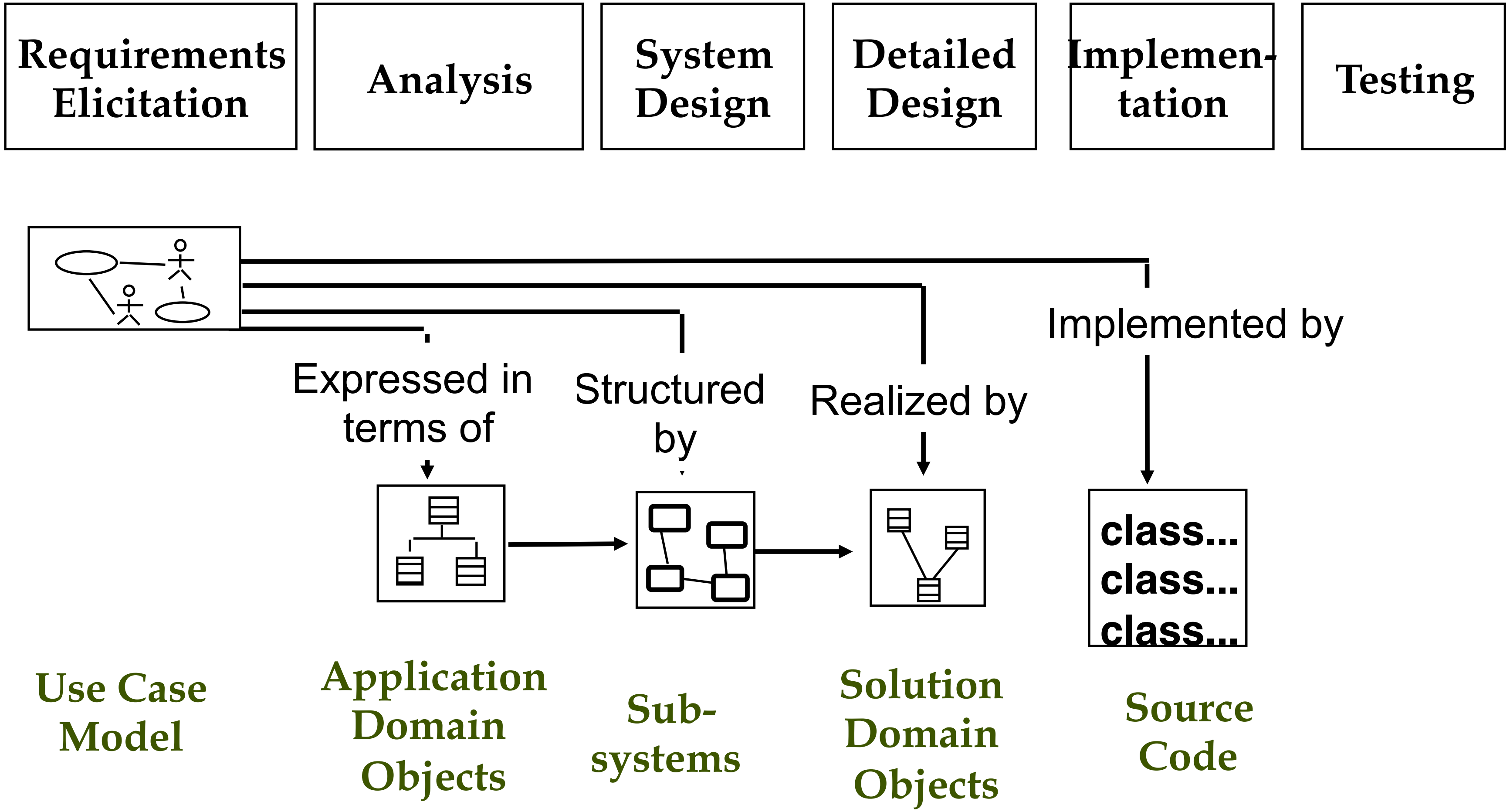
Use Case Model

Application Domain Objects

Sub-systems

Solution Domain Objects

Software Lifecycle Activities...and their models



Requirements Elicitation

Analysis

System Design

Detailed Design

Implemen-tation

Testing

Expressed in terms of

Structured by

Realized by

Implemented by

class...
class...
class...

Use Case Model

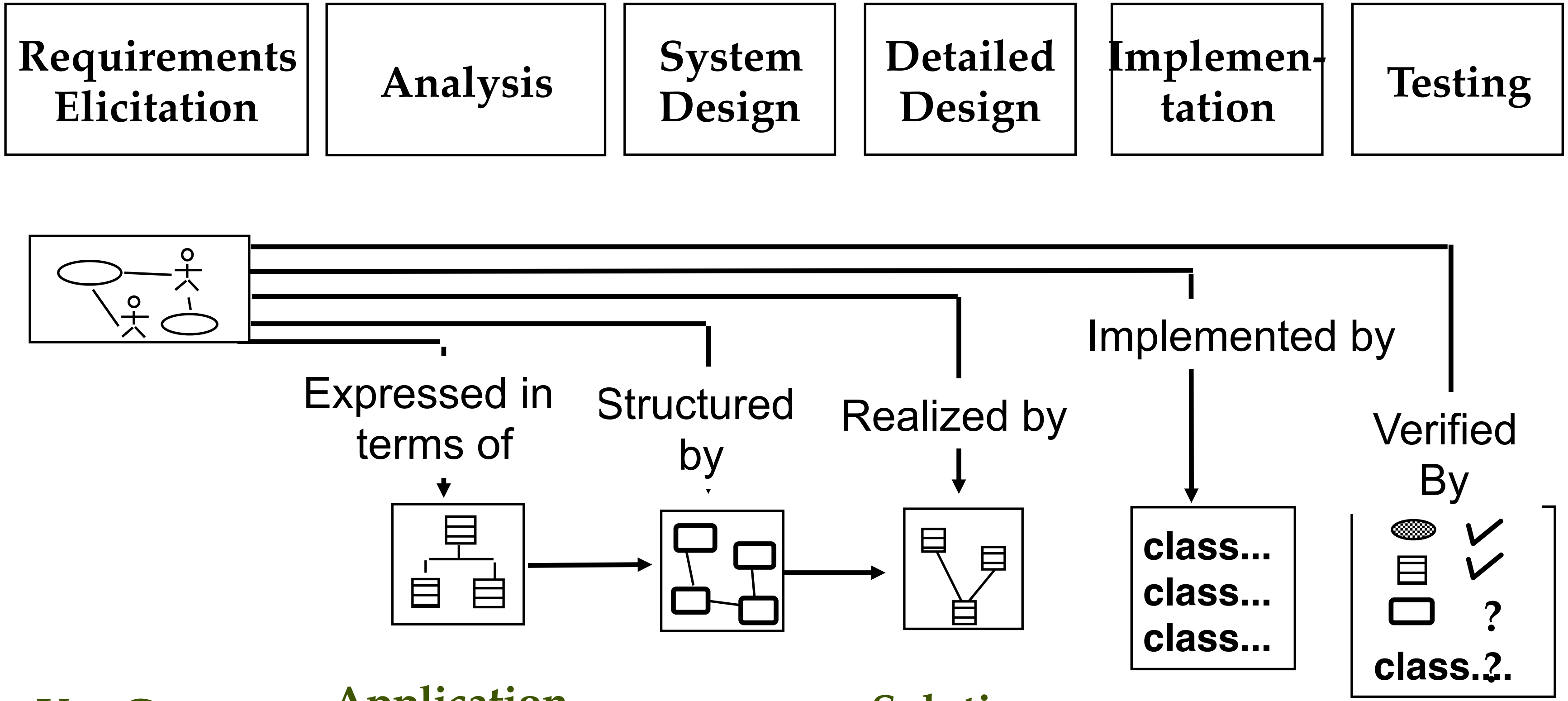
Application Domain Objects

Sub-systems

Solution Domain Objects

Source Code

Software Lifecycle Activities...and their models



Requirements Elicitation

Analysis

System Design

Detailed Design

Implemen-tation

Testing

Expressed in terms of

Structured by

Realized by

Implemented by

Verified By

class...
class...
class...

?

class...?

Use Case Model

Application Domain Objects

Sub-systems

Solution Domain Objects

Source Code

Test Case Model

Requirements Elicitation vs Analysis

- Requirements elicitation:
  - Definition of the system in terms understood by the customer and/or user ("Requirements specification")
- Analysis:
  - Definition of the system in terms understood by the developer (Technical specification, "Analysis model")

Techniques to elicit Requirements

- Bridging the gap between end user and developer:
  - **Questionnaires:** Asking the end user a list of preselected questions
  - **Task Analysis:** Observing end users in their operational environment
  - **Scenarios:** Describe the use of the system as a series of interactions between a specific 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.

- "A narrative description of what people do and experience as they try to make use of computer systems and applications"
  - [M. Carroll, Scenario-Based Design, Wiley, 1995]

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.

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..

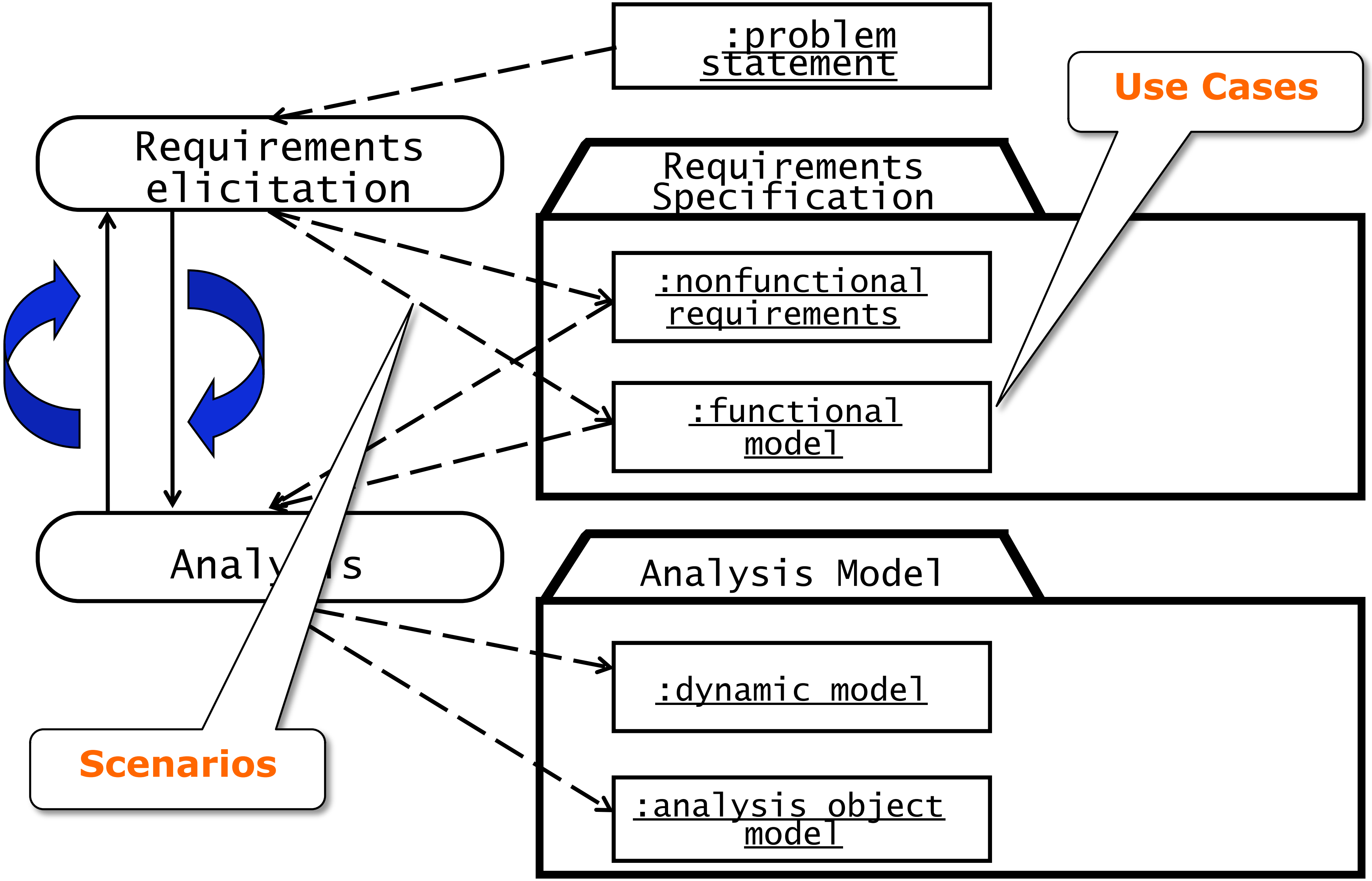
Observations about the Warehouse on Fire Scenario

- It is a concrete scenario
  - It describes a single instance of reporting a fire incident
  - It does not describe all possible situations in which a fire can be reported

- Participating actors
  - Bob, Alice and John.

From Scenarios to Use cases

Requirements Process



Requirements Specification vs Analysis Model

Both are models focusing 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

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 “
  - Also called “Pseudo requirements”.

Functional vs. Nonfunctional Requirements

Functional Requirements

- Describe user tasks which 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

Quality requirements

Constraints or
Pseudo requirements

Types of Nonfunctional Requirements (FURPS)

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

Quality requirements

Constraints or
Pseudo requirements

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

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

Some Quality Requirements Definitions

- Usability
  - The ease with which actors can perform a function in a system
  - Usability is one of the most frequently misused terms ("The system is easy to use")
  - **Usability** must be *measurable*, otherwise it is *marketing*
    - Example: Specification of the number of steps – the measure! - to perform a internet-based purchase with a web browser
- Robustness: The ability of a system to maintain a function
  - even if the user enters a wrong input
  - even if there are changes in the environment
    - Example: The system can tolerate temperatures up to 90 C
- Availability: The ratio of the expected uptime of a system to the aggregate of the expected up and down time
  - Example: The system is down not more than 5 minutes per week.

Nonfunctional Requirements: Examples

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."
  - *Usability Requirement*
- "The system must support 10 parallel tournaments"
  - *Performance Requirement*

A Task for You

- Look up the remaining definitions for the nonfunctional requirements and internalize them
    - Understand their meaning and scope (their applicability).
    - (par 4.3 of the book)
- **IMPORTANT** *(have a look by yourself)***:**
    - FURPS+ (used in Unified Process)
        - Functional, Usability, Reliability, Performance, Supportability (in ISO 9126 standard on software quality: portability, adaptability)

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.

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 component and 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!

Nonfunctional Requirements (Questions to overcome “Writers block”)

User interface and human factors

- What type of user will be using the system?
- Will more than one type of user be using the system?
- What training will be required for each type of user?
- Is it important that the system is easy to learn?
- Should users be protected from making errors?
- What input/output devices are available

Documentation

- What kind of documentation is required?
- What audience is to be addressed by each document?

Other questions reported in the book, we are skipping them because of time issues.
Study them before doing your project!!

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

Bruegge & Dutoit, 3rd edition, pp. 152