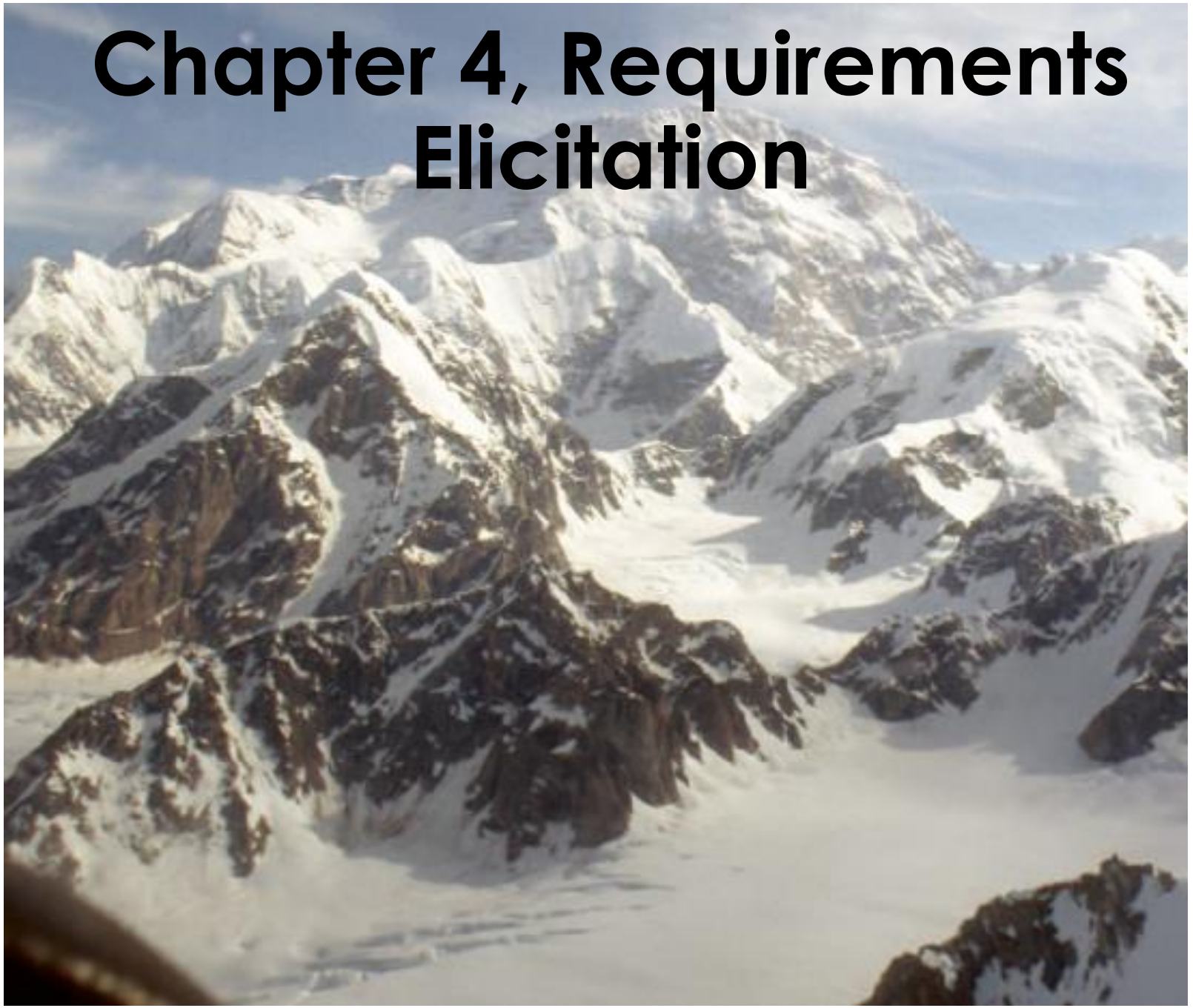
Object-Oriented Software Engineering
Using UML, Patterns, and Java

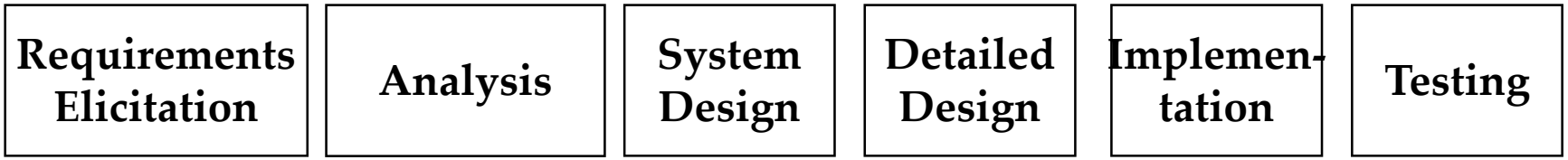
Chapter 4, Requirements Elicitation



Outline

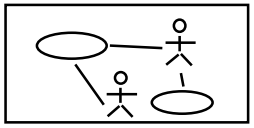
- Motivation: Software Lifecycle
- Requirements elicitation challenges
- Problem statement
- Requirements specification
  - Types of requirements
- Validating requirements
- Summary

A Typical Example of Software Lifecycle Activities



Software Lifecycle Activities...and their models

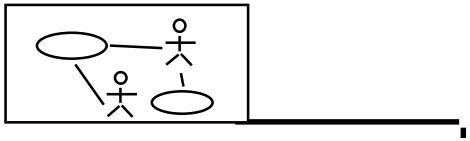
Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



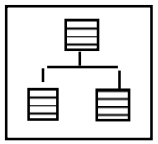
Use Case Model

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implementation	Testing



Expressed in terms of

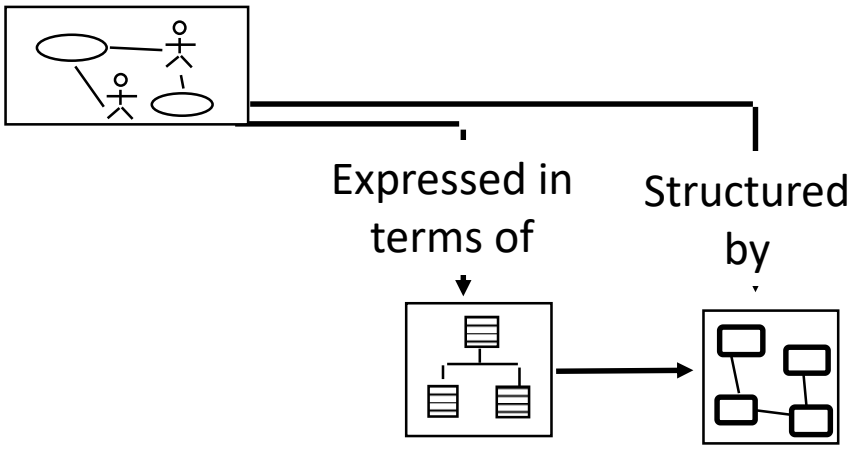


Use Case Model

Application Domain Objects

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



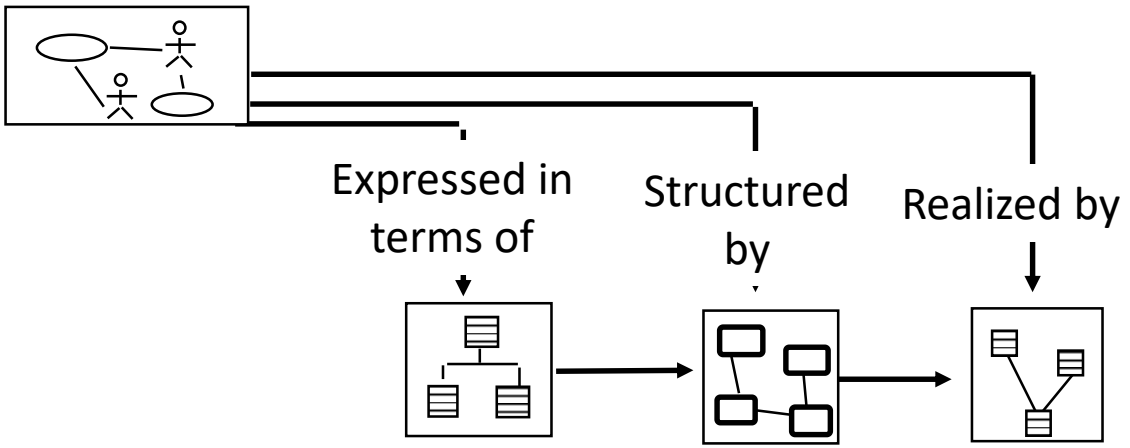
Use Case Model

Application Domain Objects

Sub-systems

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



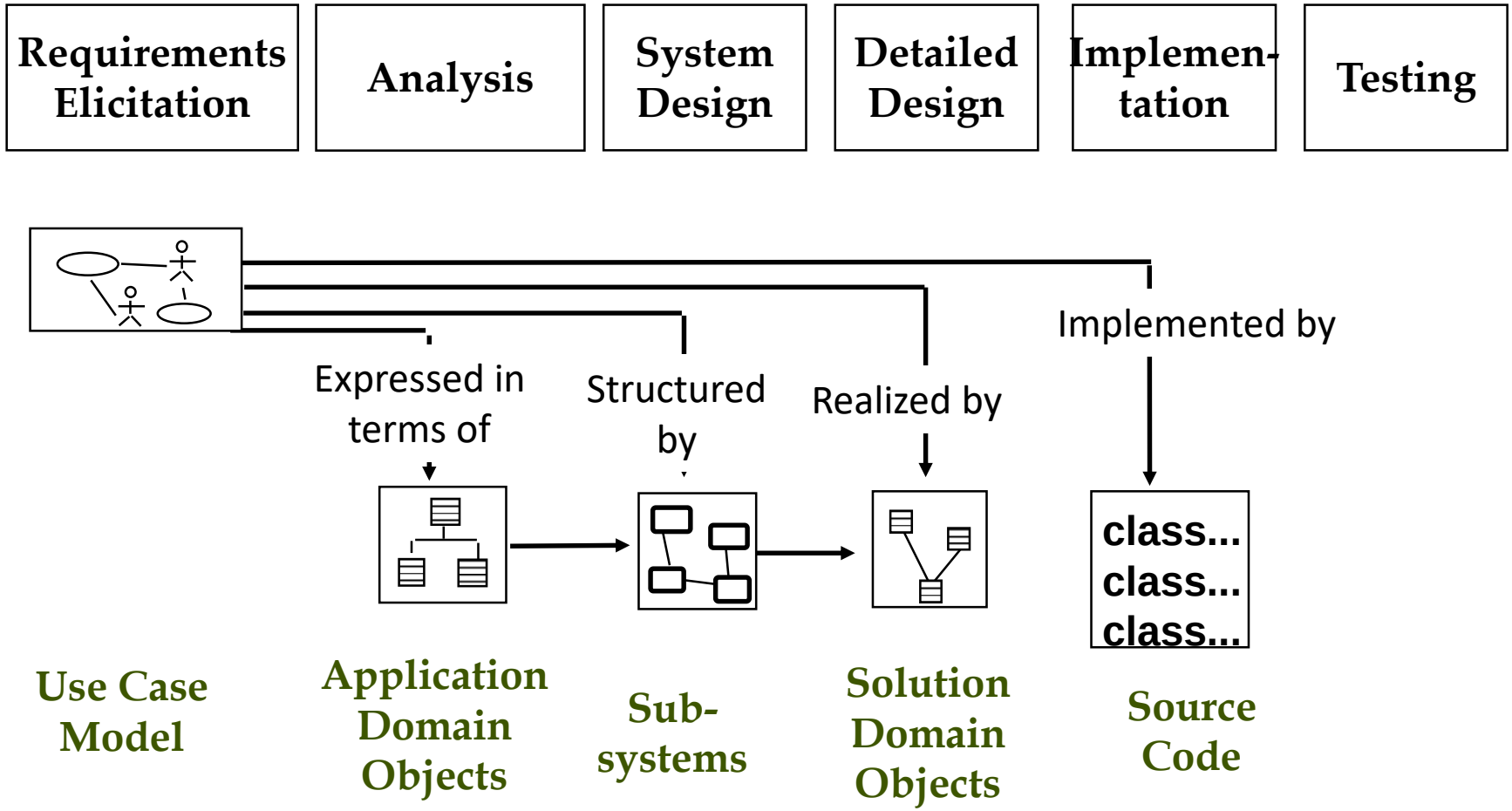
Use Case Model

Application Domain Objects

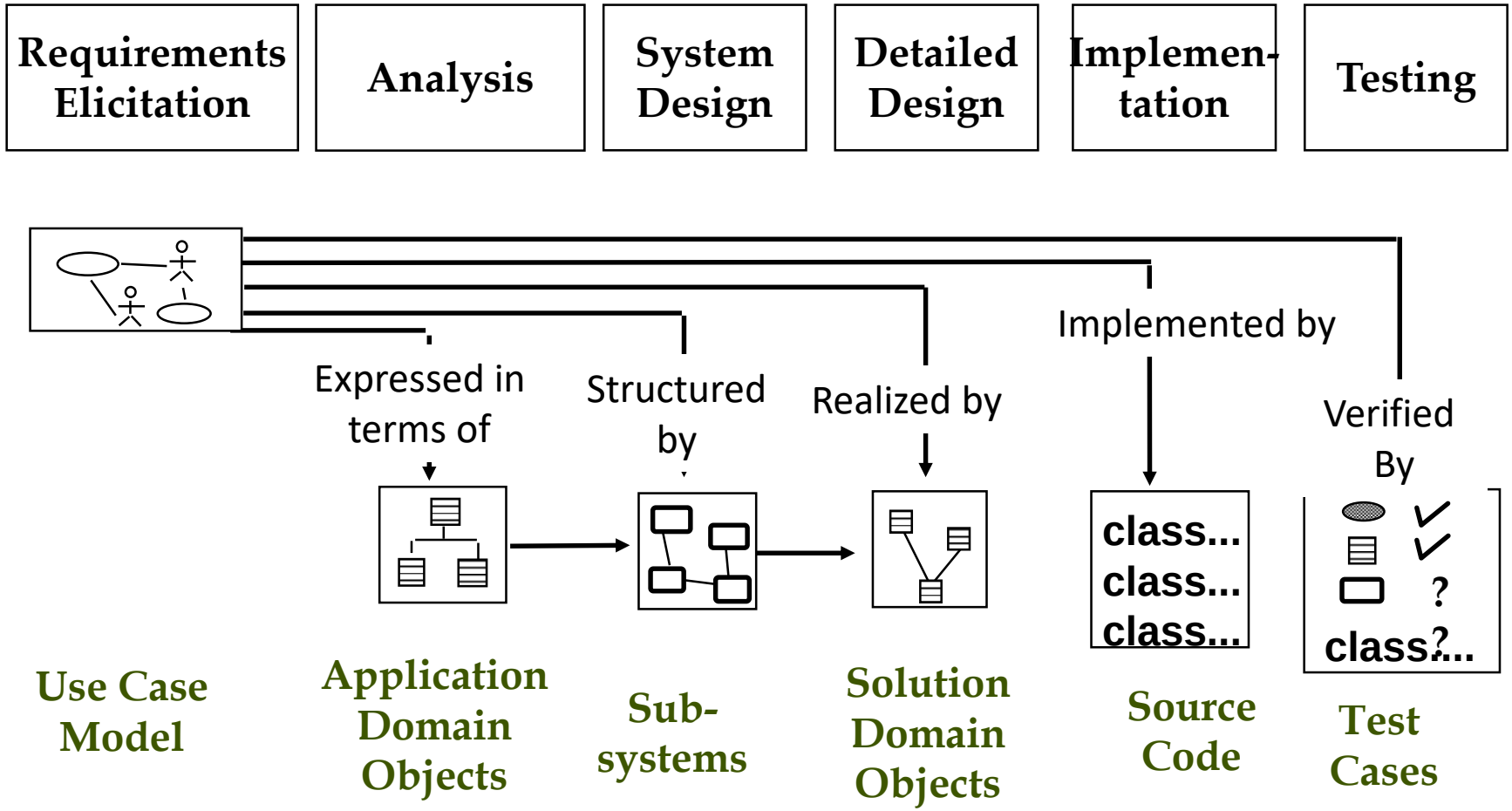
Sub-systems

Solution Domain Objects

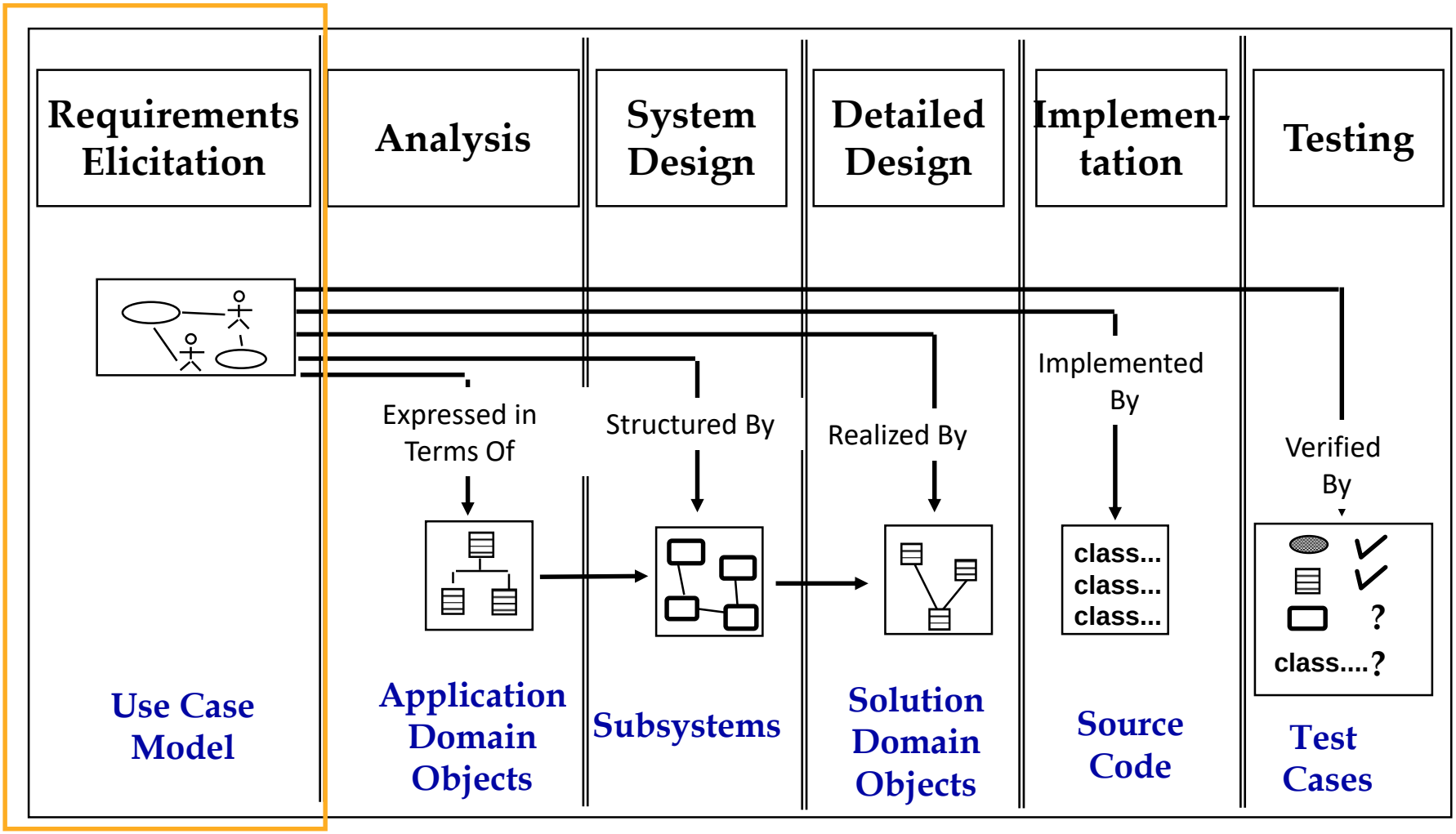
Software Lifecycle Activities...and their models



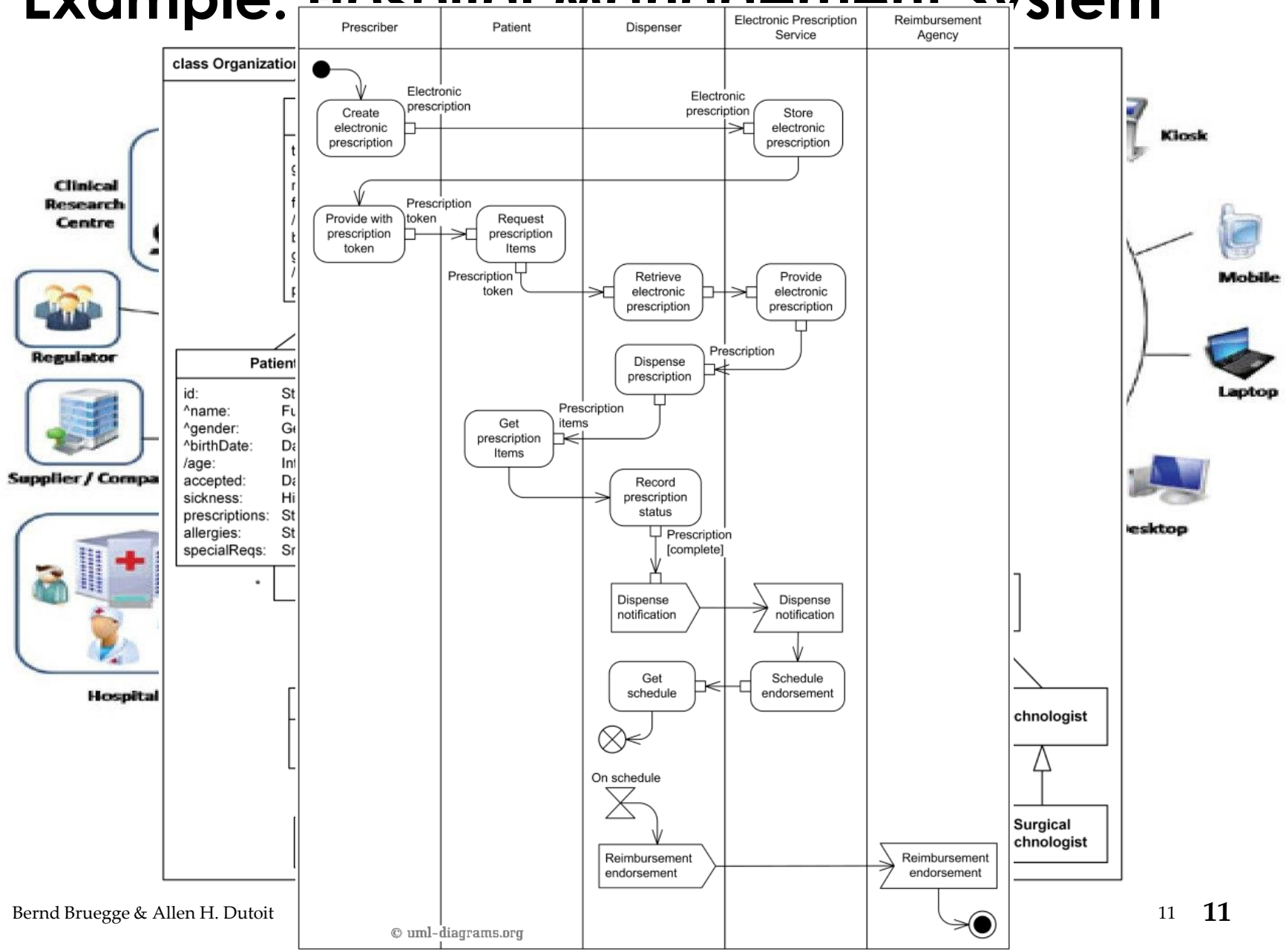
Software Lifecycle Activities...and their models



Software Lifecycle Activities



Example: Hospital Management System



What is a requirement?

- A **requirement** is a feature that the system must have or a constraint that it must satisfy to be accepted by the client.
- a thing that is needed or wanted.

Requirements Engineering

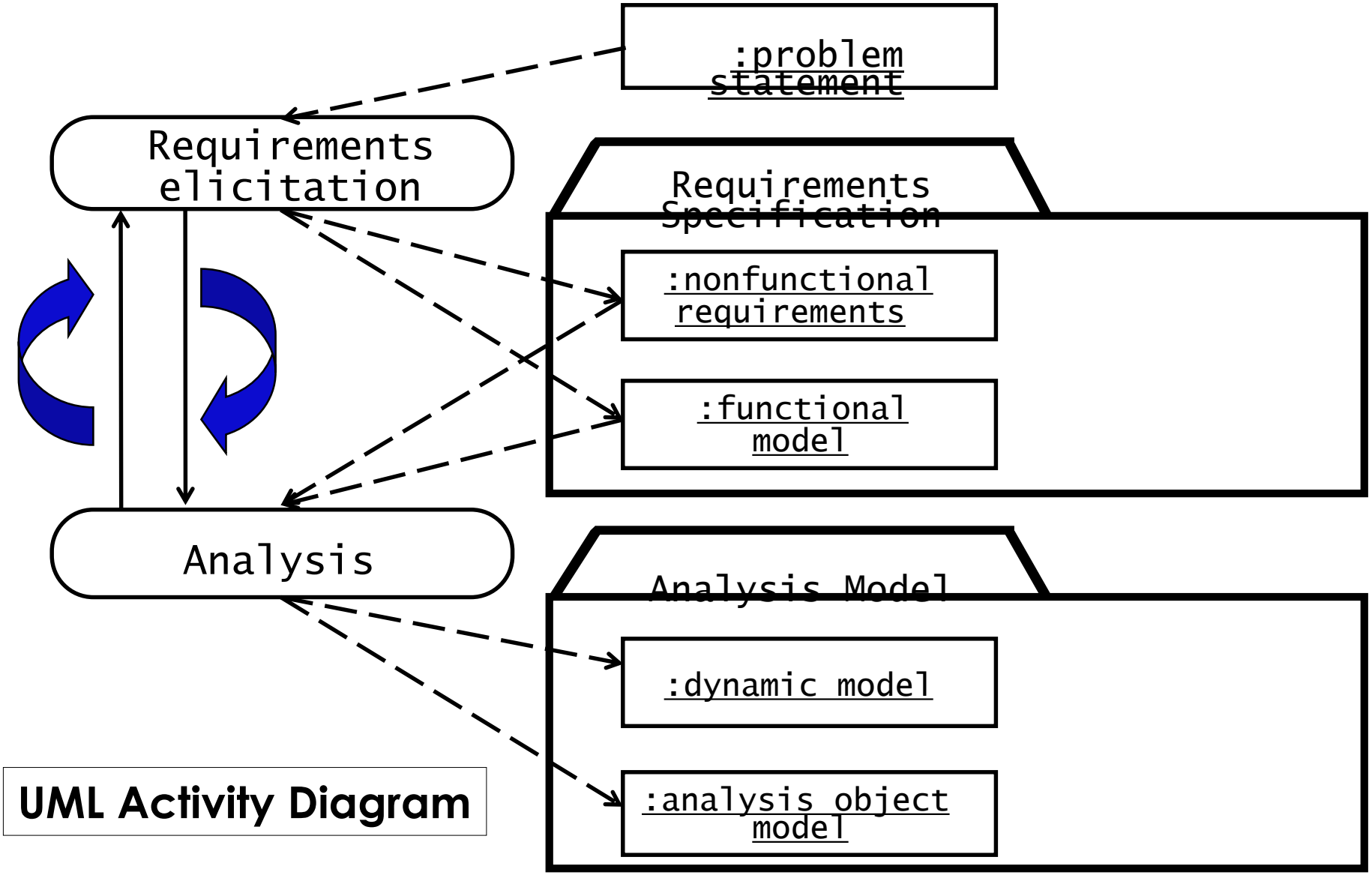
Requirements engineering aims at defining the requirements of the system under construction. Requirements engineering includes two main activities;

- *requirements elicitation*, which results in the specification of the system that the client understands, and *analysis*, which results in an analysis model that the developers can unambiguously interpret.

Requirements elicitation is the more challenging of the two because it

- requires the collaboration of several groups of participants with different backgrounds.
- On the one hand, the client and the users are experts in their domain and have a general idea of what the system should do, but they often have little experience in software development.
- On the other hand, the developers have experience in building systems, but often have little knowledge of the everyday environment of the users.

Requirements Process



UML Activity Diagram

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.

What does the Customer say?

Similarly, we often do not hear what the customers say, because we assume we know what they 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.

Scenarios

- Scenario: "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]
- A concrete, focused, informal description of a single feature of the system used by a single actor.
- Scenario vs Use Case
  - A scenario describes an example of a system in terms of a serios of interactions between the user and the system
  - A use case is an abstraction that describes a class of scenarios.
  - Both of them are written in natural language, a form that is understandable to the user

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.

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

- Scenario-based design is iterative
- Each scenario should be consisered as a work document to be augmented and rearranged ("iterated upon") when the requirements, the client acceptance criteria or the deployment situation changes.

Scenario-based Design

- Focuses on concrete descriptions and particular instances, not abstract generic ideas
- It is work driven not technology driven
- It is open-ended, it does not try to be complete
- It is informal, not formal and rigorous
- Is about envisioned outcomes, not about specified outcomes.

Types of Scenarios

- As-is scenario:
  - Describes a current situation. Usually used in re-engineering projects. The user describes the system
- 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
- Evaluation scenario:
  - Description of a user task against which the system is to be evaluated.
- Training scenario:
  - A description of the step by step instructions that guide a novice user through a system

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.

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
- Functional Modeling (see next lecture)

Requirement Elicitation Activities

- Identifying Actors (Section 4.4.1)
- Identifying Scenarios (Section 4.4.2)
- Identifying Use Cases (Section 4.4.3)
- Refining Use Cases (Section 4.4.4)
- Identifying Relationships Among Actors and Use Cases (Section 4.4.5)
- Identifying Initial Analysis Objects (Section 4.4.6)
- Identifying Nonfunctional Requirements (Section 4.4.7).

1. Questions for identifying actors

- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions, such as maintenance and administration?
- With what external hardware or software system will the system interact?

2. Questions for identifying scenarios

- What are the tasks that the actor wants the system to perform?
- What information does the actor access? Who creates that data? Can it be modified or removed? By whom?
- Which external changes does the actor need to inform the system about? How often? When?
- Which events does the system need to inform the actor about? With what latency?

3. Use Cases

- Use cases should be named with verb phrases. The name of the use case should indicate what the user is trying to accomplish (e.g., ReportEmergency, OpenIncident).
- Actors should be named with noun phrases (e.g., FieldOfficer, Dispatcher, Victim).
- The boundary of the system should be clear. Steps accomplished by the actor and steps accomplished by the system should be distinguished
- Use case steps in the flow of events should be phrased in the active voice. This makes it explicit who accomplished the step.
- A use case should describe a complete user transaction (e.g., the ReportEmergency use case describes all the steps between initiating the emergency reporting and receiving an acknowledgment).
- Exceptions should be described separately.
- A use case should not describe the user interface of the system. This takes away the focus from the actual steps accomplished by the user and is better addressed with visual mock-ups (e.g., refer to the function, not the menu, the button).
- A use case should not exceed two or three pages in length. Otherwise, use include

4. Refining use cases

- First, refine a single scenario to understand the user's assumptions about the system. The user may be familiar with similar systems, in which case, adopting specific user interface conventions would make the system more usable.
- Next, define many not-very-detailed scenarios to define the scope of the system. Validate with the user.
- Use mock-ups as visual support only; user interface design should occur as a separate task after the functionality is sufficiently stable.
- Present the user with multiple and very different alternatives (as opposed to extracting a single alternative from the user). Evaluating different alternatives broadens the user's horizon. Generating different alternatives forces developers to "think outside the box."

5. Identifying Relationships among actors and Use Cases

- Use extend relationships for exceptional, optional, or seldom-occurring behavior. An example of seldom-occurring behavior is the breakdown of a resource (e.g., a fire truck). An example of optional behavior is the notification of nearby resources responding to an unrelated incident.
- Use include relationships for behavior that is shared across two or more use cases.
- However, use discretion when applying the above two heuristics and do not overstructure the use case model. A few longer use cases (e.g., two pages long) are easier to understand and review than many short ones (e.g., ten lines long).

6. Identifying Analysis Objects

- Terms that developers or users must clarify to understand the use case
- Recurring nouns in the use cases (e.g., Incident)
- Real-world entities that the system must track (e.g., FieldOfficer, Resource)
- Real-world processes that the system must track (e.g., EmergencyOperationsPlan)
- Use cases (e.g., ReportEmergency)
- Data sources or sinks (e.g., Printer)
- Artifacts with which the user interacts (e.g., Station)
- *Always* use application domain terms.

7. Identify Non-Functional requirements

- More about it later…

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

Example of an Unintended Feature

From the News: London underground train leaves station without driver!

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him
  - The driver door was not treated as a door in the source code!

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

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

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

Example

Quality requirements for SatWatch

- Any user who knows how to read a digital watch and understands international time zone abbreviations should be able to use SatWatch without the user manual. [Usability requirement]
- As the SatWatch has no buttons, no software faults requiring the resetting of the watch should occur. [Reliability requirement]
- SatWatch should display the correct time zone within 5 minutes of the end of a GPS blackout period. [Performance requirement]
- SatWatch should measure time within 1/100th second over 5 years. [Performance requirement]
- SatWatch should display time correctly in all 24 time zones. [Performance requirement]
- SatWatch should accept upgrades to its onboard via the Webify Watch serial interface. [Supportability requirement]

Constraints for SatWatch

- All related software associated with SatWatch, including the onboard software, will be written using Java, to comply with current company policy. [Implementation requirement]
- SatWatch complies with the physical, electrical, and software interfaces defined by WebifyWatch API 2.0. [Interface requirement]

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 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!

We can specify Requirements for "Requirements Management"

- Functional requirements:
  - Store the requirements in a shared repository
  - Provide multi-user access to the requirements
  - Automatically create a specification document from the requirements
  - Allow change management of the requirements
  - Provide traceability of the requirements throughout the artifacts of the system.

Tools for Requirements Management (2)

DOORS (Telelogic)

- Multi-platform requirements management tool, for teams working in the same geographical location. DOORS XT for distributed teams

RequisitePro (IBM/Rational)

- Integration with MS Word
- Project-to-project comparisons via XML baselines

RD-Link (http://www.ring-zero.com)

- Provides traceability between RequisitePro & Telelogic DOORS

Unicase (http://unicase.org)

- Research tool for the collaborative development of system models
- Participants can be geographically distributed.

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

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

Section 3.3 Nonfunctional Requirements

3.3.1 User interface and human factors
3.3.2 Documentation
3.3.3 Hardware considerations
3.3.4 Performance characteristics
3.3.5 Error handling and extreme conditions
3.3.6 System interfacing
3.3.7 Quality issues
3.3.8 System modifications
3.3.9 Physical environment
3.3.10 Security issues
3.3.11 Resources and management issues

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?

Nonfunctional Requirements (2)

Hardware considerations

- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Performance characteristics

- Are there speed, throughput, response time constraints on the system?
- Are there size or capacity constraints on the data to be processed by the system?

Error handling and extreme conditions

- How should the system respond to input errors?
- How should the system respond to extreme conditions?

Nonfunctional Requirements (3)

System interfacing

- Is input coming from systems outside the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?

Quality issues

- What are the requirements for reliability?
- Must the system trap faults?
- What is the time for restarting the system after a failure?
- Is there an acceptable downtime per 24-hour period?
- Is it important that the system be portable?

Nonfunctional Requirements (4)

System Modifications

- What parts of the system are likely to be modified?
- What sorts of modifications are expected?

Physical Environment

- Where will the target equipment operate?
- Is the target equipment in one or several locations?
- Will the environmental conditions be ordinary?

Security Issues

- Must access to data or the system be controlled?
- Is physical security an issue?

Nonfunctional Requirements (5)

Resources and Management Issues

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?