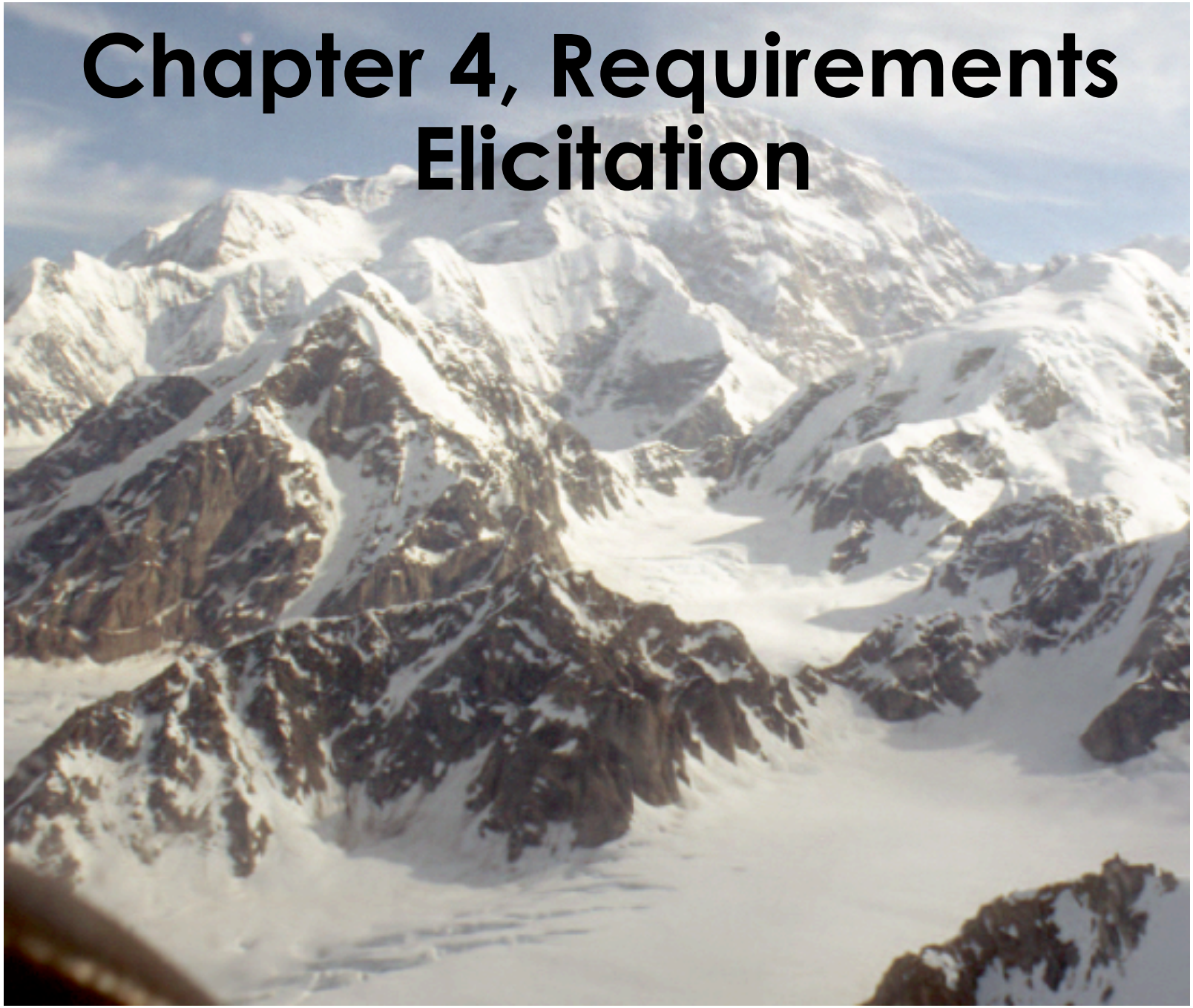
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 4, Requirements Elicitation



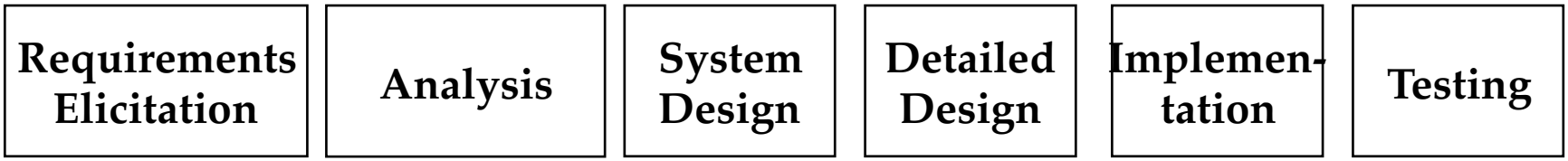
Outline

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

Software Lifecycle Definition

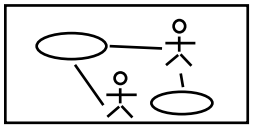
- Software lifecycle
  - Models for the development of software
    - Set of activities and their dependency relationships to each other to support the development of a software system
    - Examples:
      - Analysis, Design, Implementation, Testing
- Typical Lifecycle questions:
  - Which activities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

A Typical Example of Software Lifecycle Activities



Software Lifecycle Activities...and their models

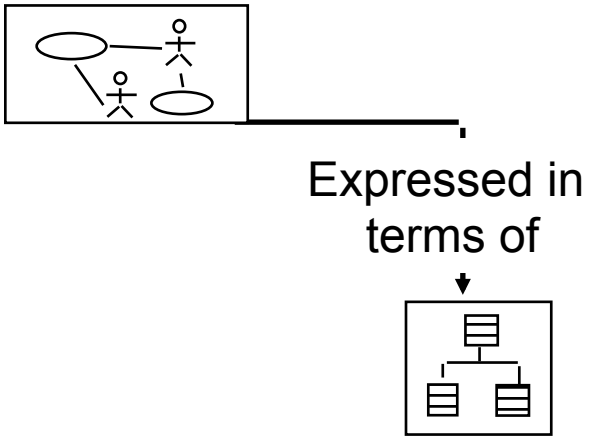
Requirements Elicitation	Analysis	System Design	Detailed Design	Implementation	Testing



Use Case Model

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implementation	Testing

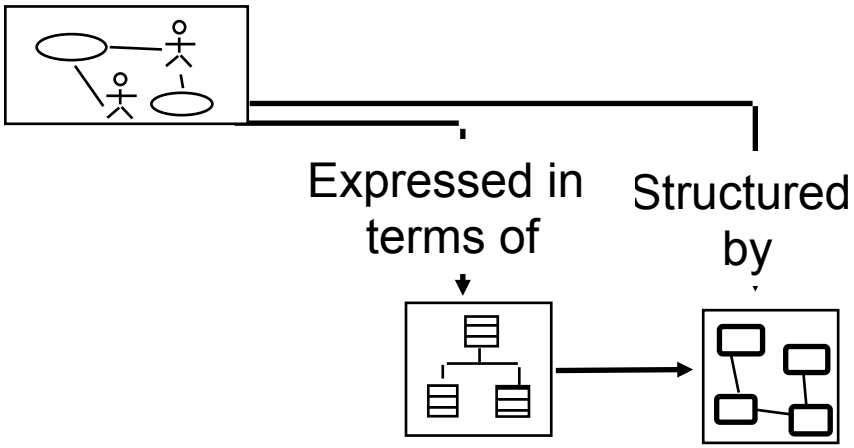


Use Case Model

Application Domain Objects

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



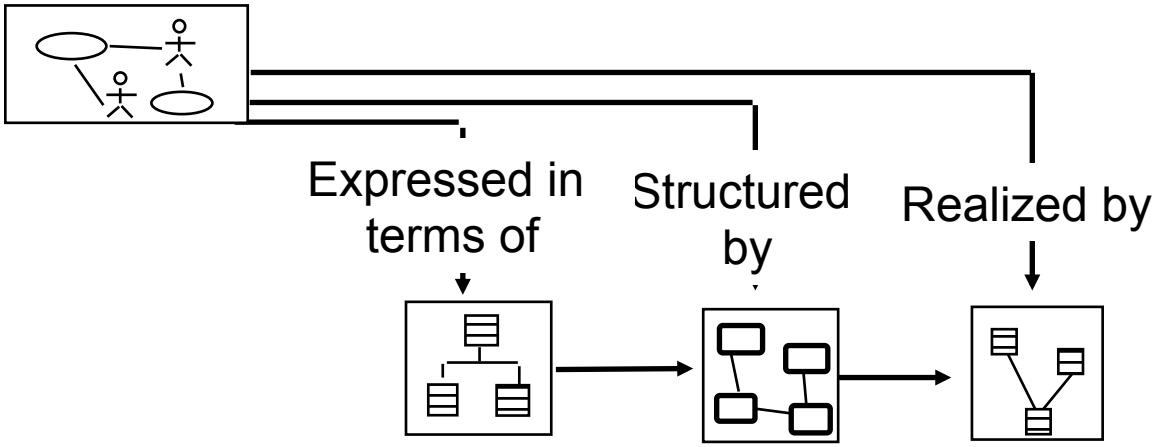
Use Case Model

Application Domain Objects

Sub-systems

Software Lifecycle Activities...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



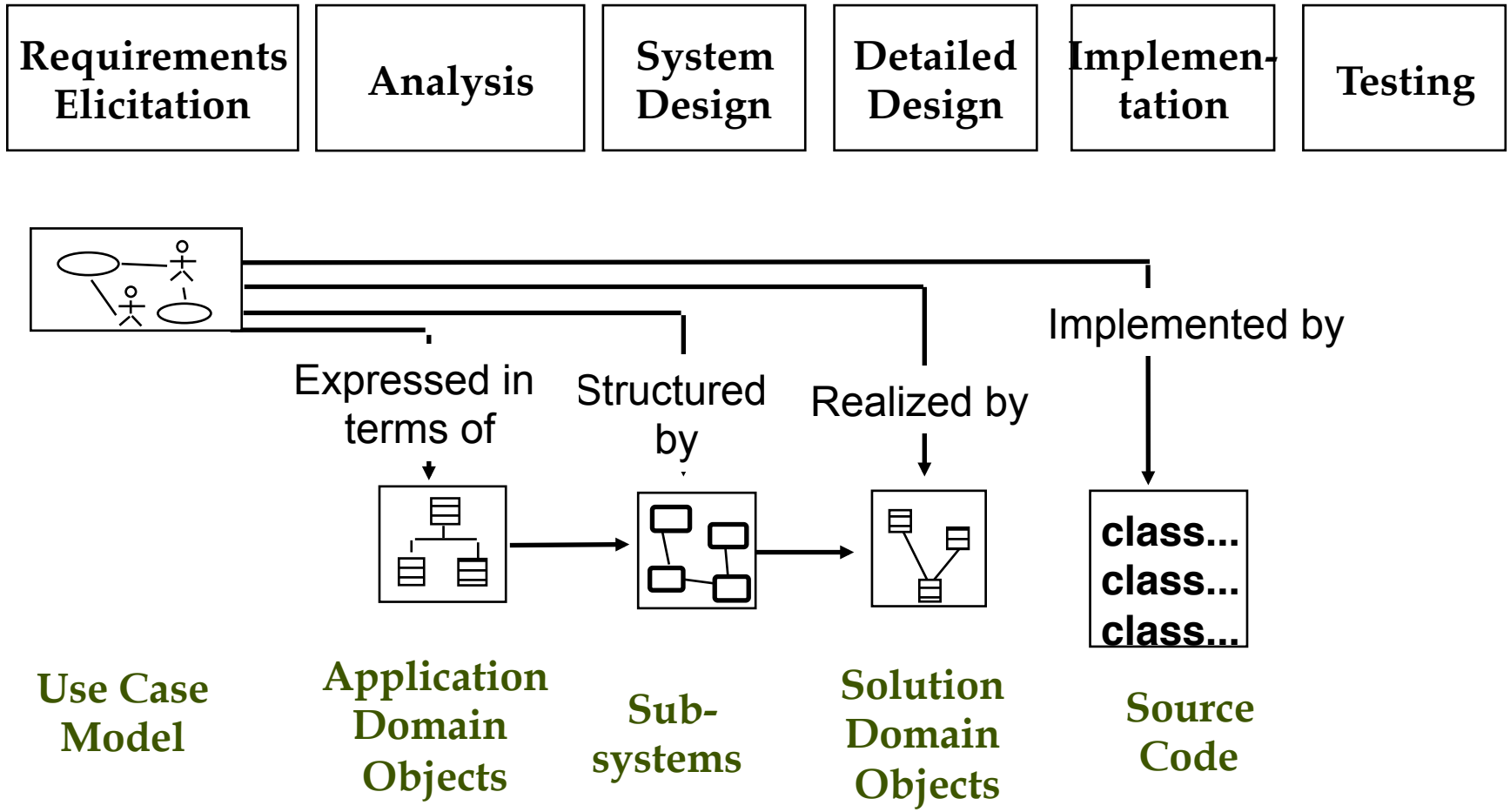
Use Case Model

Application Domain Objects

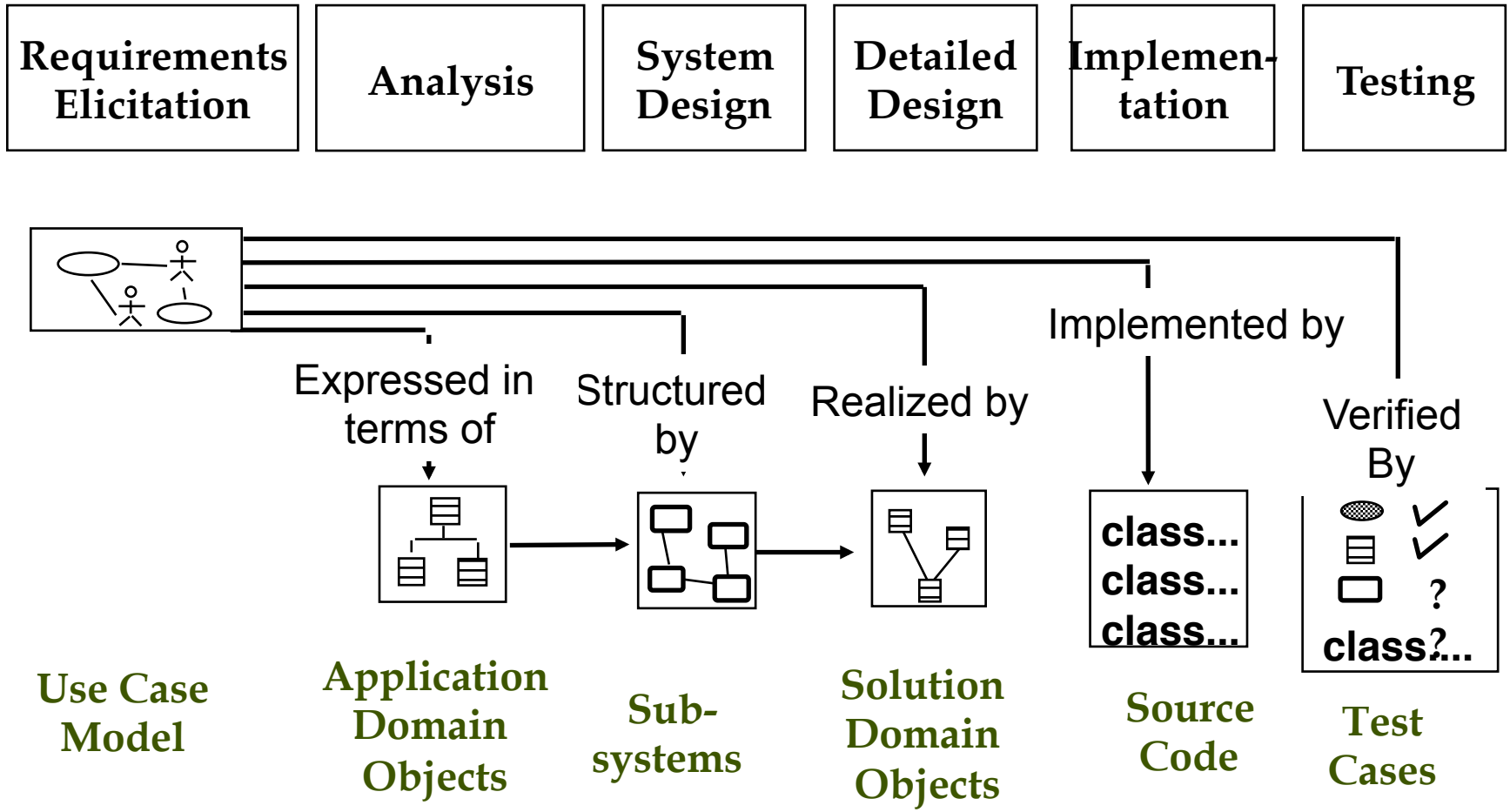
Sub-systems

Solution Domain Objects

Software Lifecycle Activities...and their models



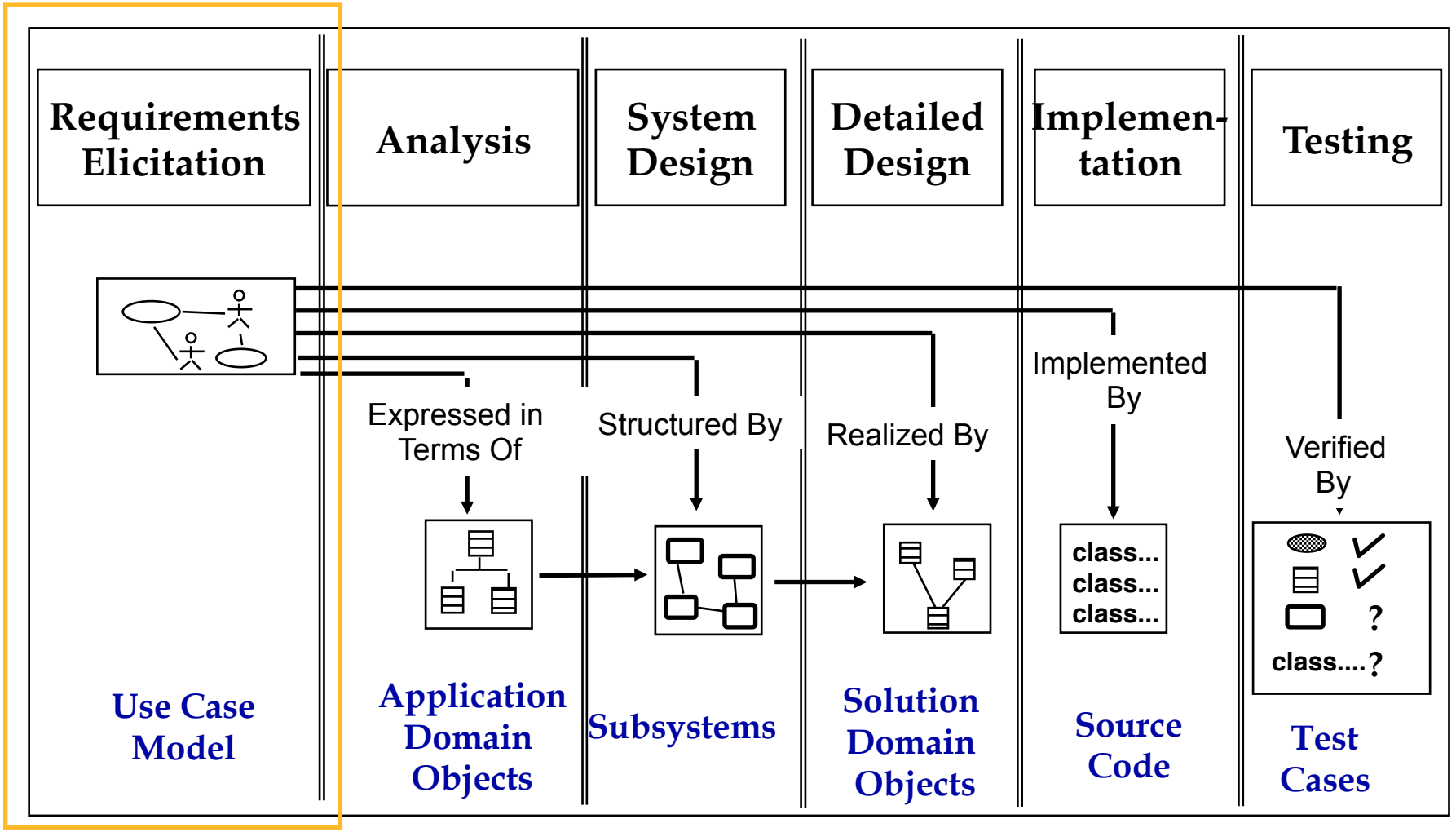
Software Lifecycle Activities...and their models



What is the best Software Lifecycle?

- Answering this question is the topics of the lecture on software lifecycle modeling
- For now we assume we have a set of predefined activities:
  - Today we focus on the activity requirements elicitation

Software Lifecycle Activities



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.

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 (Italian: that which is pinned to the scenery)
  - 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.

More Definitions

- 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-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.

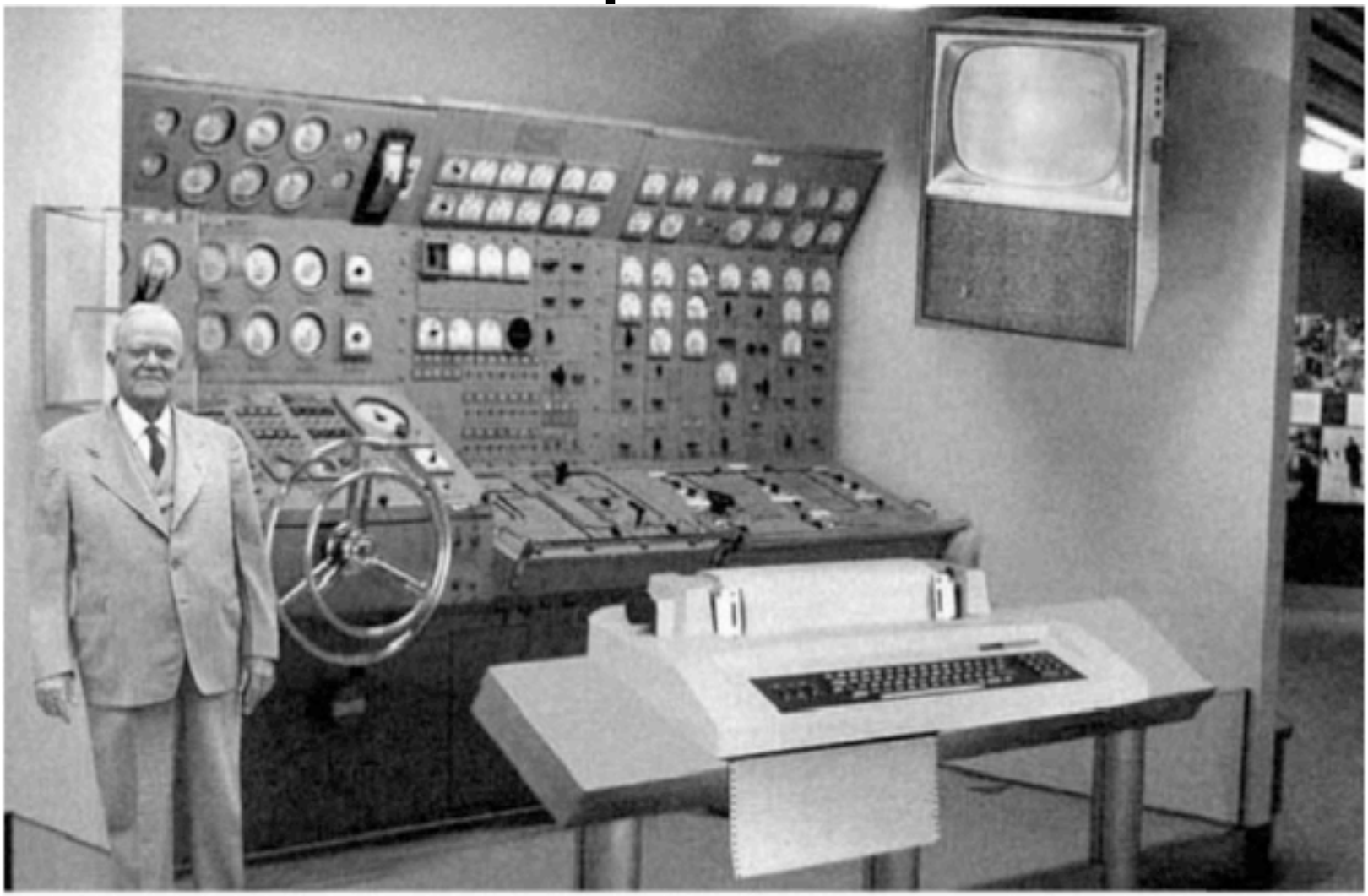
Scenario-based Design

- 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
    - 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.

A Visionary Scenario (1954): The Home Computer in 2004



Scientists from the **RAND** *Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.*

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.

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)

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

=> 3 Examples.

Defining the System Boundary is difficult

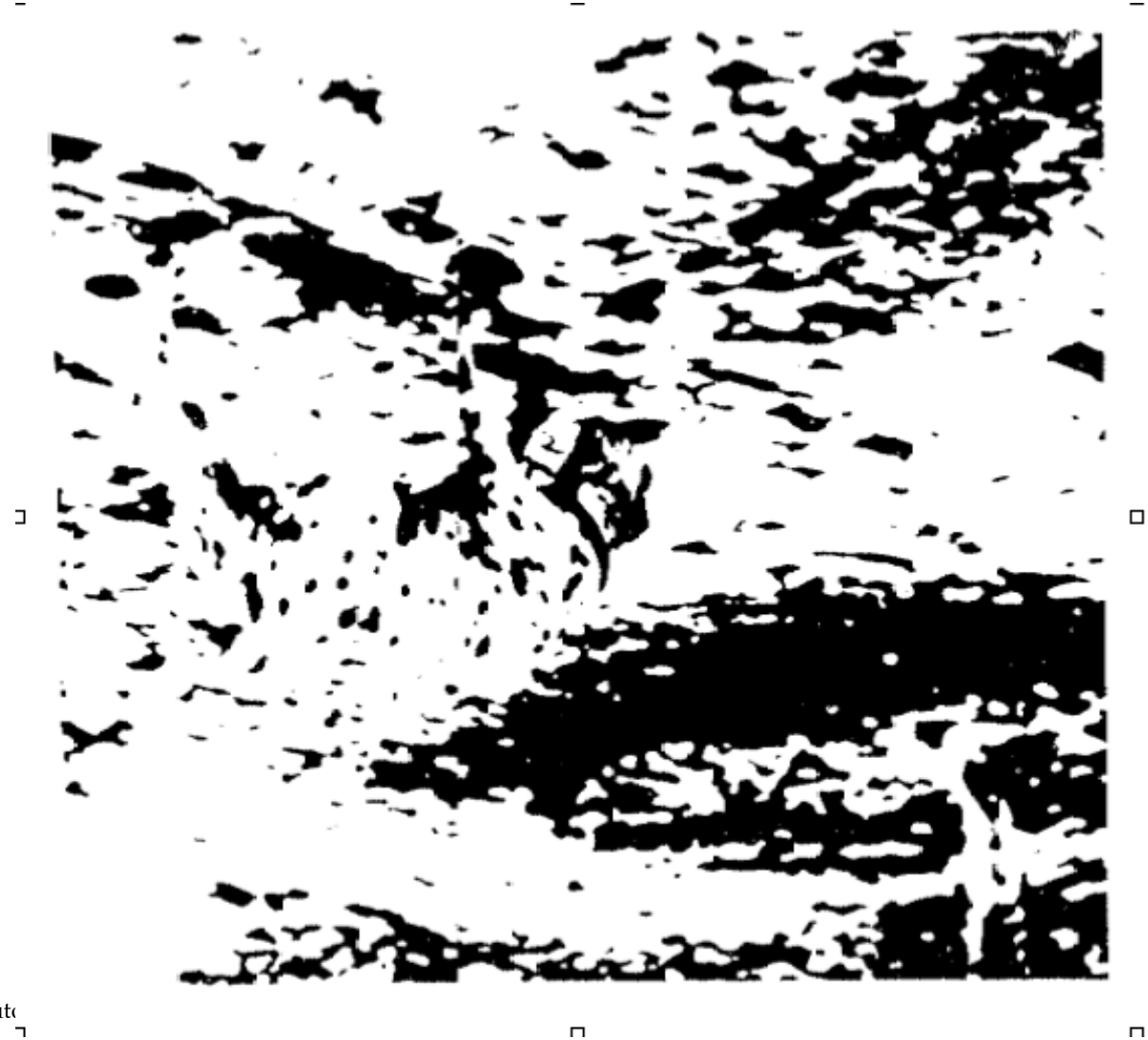
What do you see here?

Defining the System Boundary is difficult

What do you see now?

Defining the System Boundary is difficult

What do you see now?



Example of an Ambiguous Specification

During a laser experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as “10023”

The light beam never hit the mountain top
What was the problem?

The computer interpreted the number in miles...

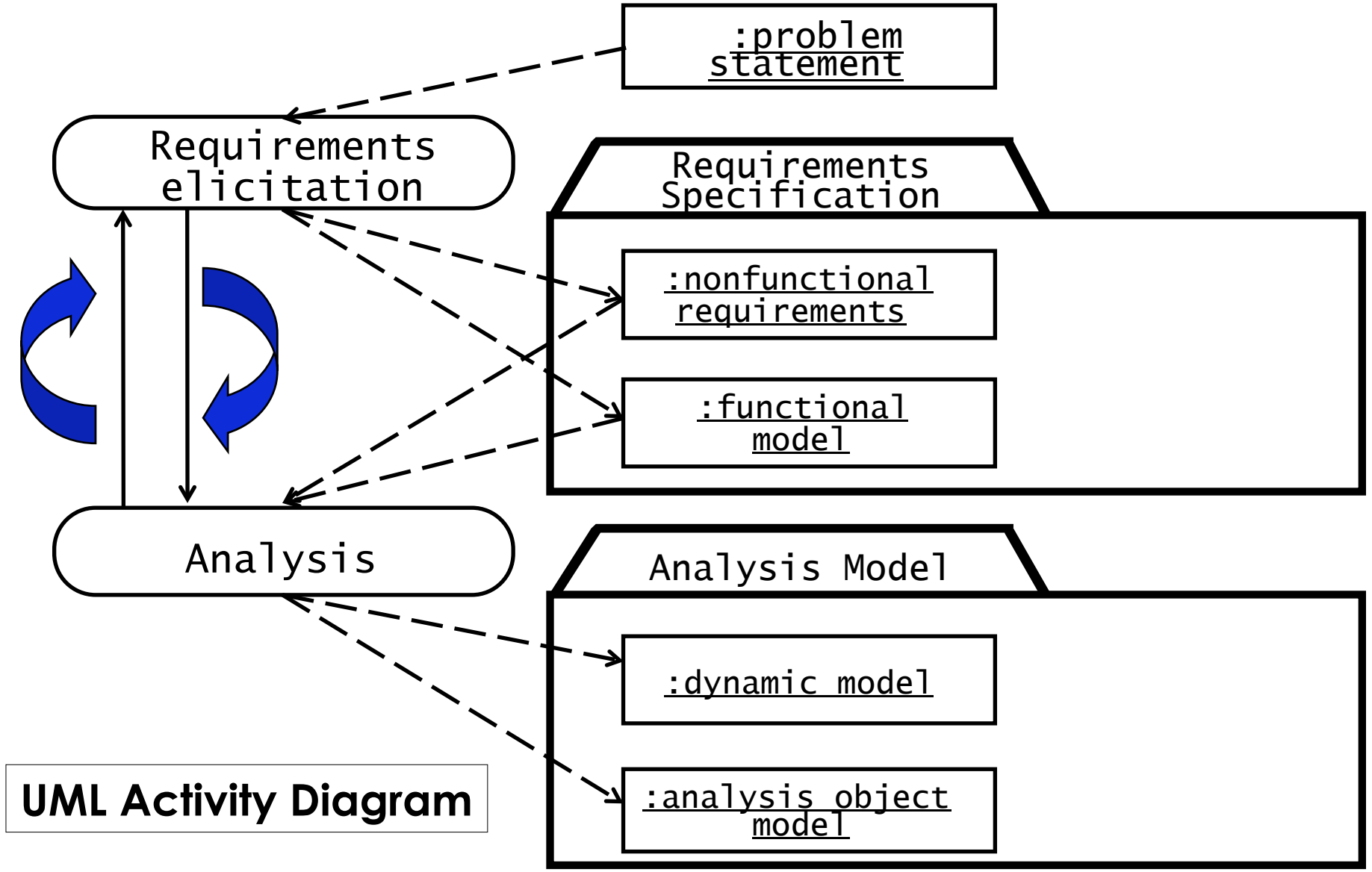
Example of an Unintended Feature

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!

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

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

Task

- Find definitions for all the nonfunctional requirements on the previous slide and learn them by heart
- Understand their meaning and scope (their applicability).

Some Quality Requirements Definitions

- Usability
  - The ease with which actors can use a system to perform a function
  - 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*
- "The operator must be able to add new games without modifications to the existing system."
  - *Supportability Requirement*

What should not be in the Requirements?

- System structure, implementation technology
- Development methodology
  - Parnas, How to fake the software development process
- 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?

Example: Heathrow Luggage System

- On April 5, 2008 a system update was performed to upgrade the baggage handling:
  - 50 flights were canceled on the day of the update
  - A “Bag Backlog” of 20,000 bags was produced (Naomi Campbell had a fit and was arrested)
  - The bags were resorted in Italy and eventually sent to the passengers via Federal Express
- What happened? Initial explanation:
  - Computer failure in the high storage bay area in combination with shortage of personal

Exercise

- Reverse engineer the requirements for the Heathrow luggage system
  - Use the requirements analysis document template
  - Use available information on the internet.
- Questions to ask:
  - How are the bags stored after passengers have checked, but before they enter the plane?
  - How are the bags retrieved from the storage area?
  - What about existing luggage systems (“legacy systems”)?
  - Scalability: How many users should the new luggage system support? How can this be tested before deployment?
  - Throughput: How many suit cases/hour need to be supported?

Bonus Question

- What changes to the requirements should have been done to avoid the Heathrow desaster?

Exercise Solution

- Available information on the internet. Examples
  - http://blogs.zdnet.com/projectfailures/?p=610
  - http://www.bloomberg.com/apps/news?pid=conewsstory&refer=conews&tkr=FDX:US&sid=aY4IqhBRcytA
- Examples of requirements:
  - Automate the processing of No-Show passengers
  - Use a high bay storage area ("high rack warehouse")
  - Provide a chaotic storage capability
  - Combine two existing luggage systems ("legacy systems"): Early (hours before) and last minute checkins
  - The system must be tested with 2500 volunteers
  - The throughput must be at least 12000 suitcases/hour.

Additional Readings

- Scenario-Based Design
  - John M. Carrol, Scenario-Based Design: Envisioning Work and Technology in System Development, John Wiley, 1995
  - Usability Engineering: Scenario-Based Development of Human Computer Interaction, Morgan Kaufman, 2001
- Heathrow Luggage System:
  - http://blogs.zdnet.com/projectfailures/?p=610
  - http://www.bloomberg.com/apps/news?pid=conewsstory&refer=conews&tkr=FDX:US&sid=aY4IqhBRcytA

Additional Information about Heathrow (In German)

Panne auf Flughöhe Null (Spiegel):
http://www.spiegel.de/reise/aktuell/0,1518,544768,00.html

Zurück in das rotierende Chaos (FAZ):
http://www.faz.net/s/Rub7F4BEE0E0C39429A8565089709B70C44/Doc~EC1120B27386C4E34A67A5EE8E5523433~ATpl~Ecommon~Scontent.html