Software Engineering
Requirements Elicitation

Prof. Dr. Peter Müller

Software Component Technology

The slides in this section are partly based on the lecture
“Software Engineering I” by Prof. Bernd Brügge, TU München

Summer Semester 06

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

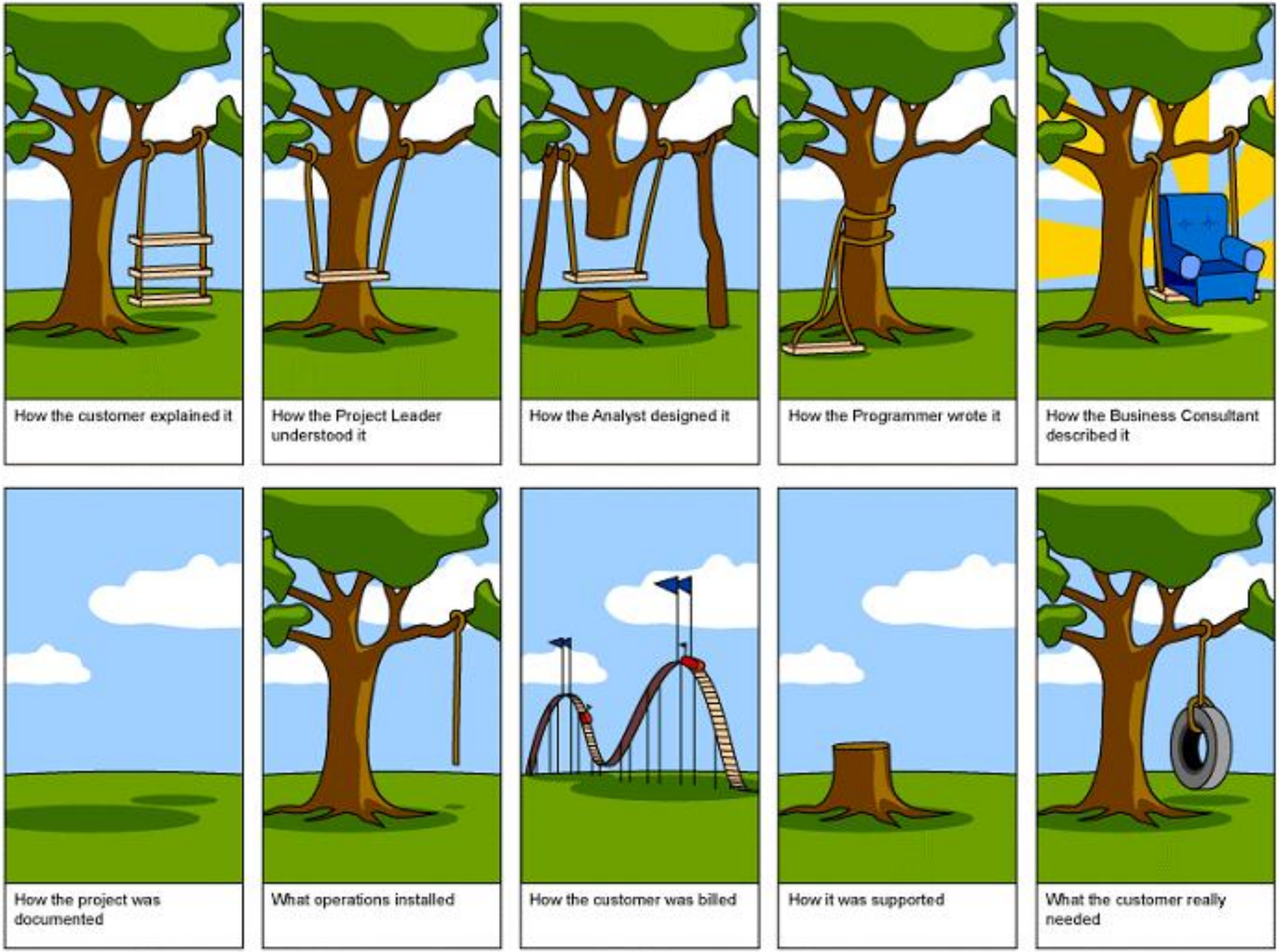
2. Requirements Elicitation

2.1 Requirements

2.2 Documenting Functional Requirements

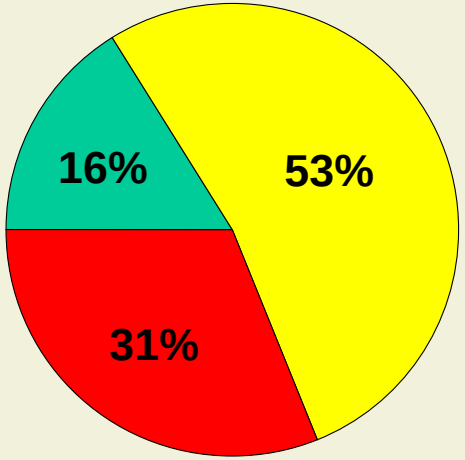
2.3 Requirements Elicitation Activities

2.4 Requirements Documentation


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

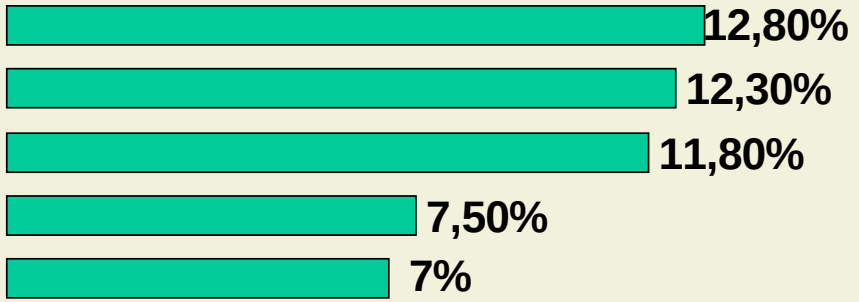
Software – a Poor Track Record

- Software bugs cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product
- 84% of all software projects are unsuccessful
  - Late, over budget, less features than specified, cancelled
- The average unsuccessful project
  - 222% longer than planned
  - 189% over budget
  - 61% of originally specified features

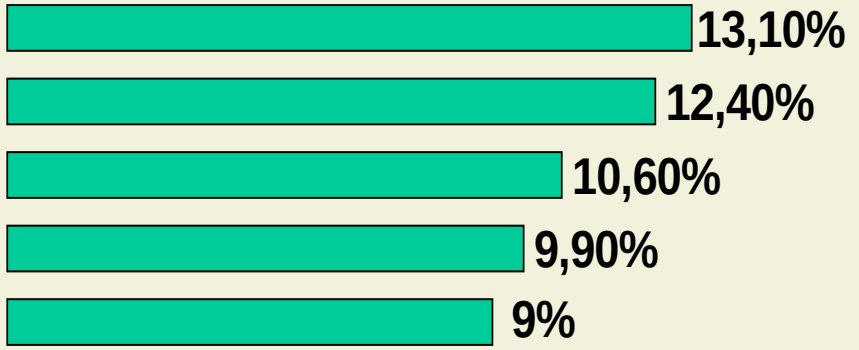


Why IT-Projects Fail

- Top 5 reasons measured by frequency of responses by IT executive management
- Failure profiles of yellow projects
  1. **Lack of User Input**
  2. **Incomplete Requirements**
  3. **Changing Requirements**
  4. Lack of Executive Support
  5. Technology Incompetence



- Failure profiles of red projects
  1. **Incomplete Requirements**
  2. **Lack of User Involvement**
  3. Lack of Resources
  4. **Unrealistic Expectations**
  5. Lack of Executive Support



2. Requirements Elicitation

Requirements

- Definition:
  A feature that the system must have or a constraint it must satisfy to be accepted by the client

  [Brügge, Dutoit]

- **Requirements engineering** defines the requirements of the system under construction

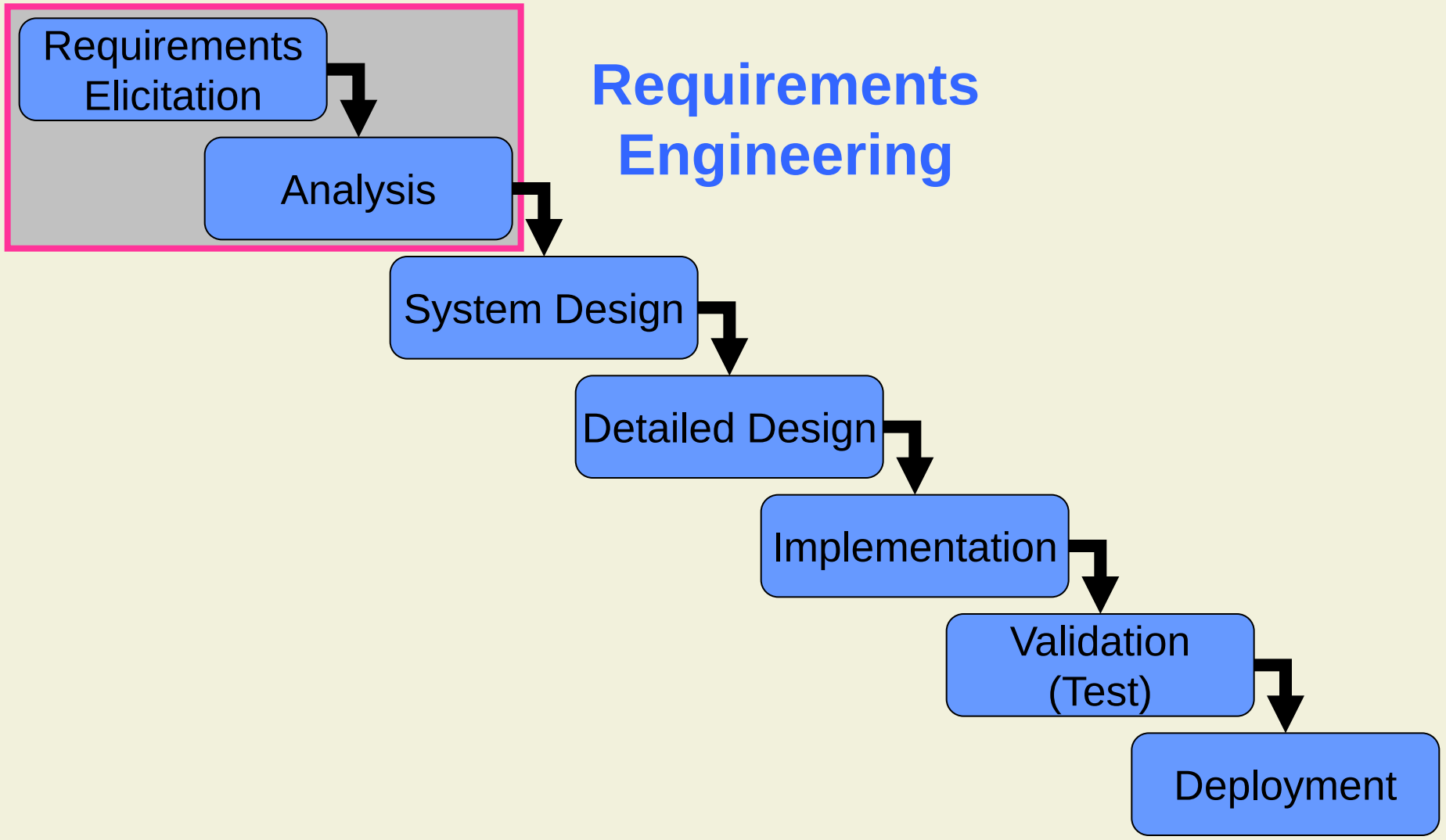
Requirements

- Describe the **user's view** of the system
- Identify the **what** of the system, not the **how**

- **Part of requirements**
  - Functionality
  - User interaction
  - Error handling
  - Environmental conditions (interfaces)

- **Not part of requirements**
  - System structure
  - Implementation technology
  - System design
  - Development methodology

Waterfall Model of Project Life Cycle



Requirements Engineering: Overview

Client

Users

Requirements Elicitation

Requirements specification

Analysis

Analysis Model

Design

Designers

Participation

Used for communication

Requirements Elicitation vs. Analysis

- Requirements specification and analysis model **represent the same information**

- **Requirements Elicitation**
  - Definition of the system in terms **understood by the customer**
  - Requirements specification uses **natural language**
  - Communication with **clients and users**

- **Analysis**
  - Technical specification of the system in terms **understood by the developer**
  - The analysis model uses a **formal or semi-formal notation**
  - Communication among **developers**

Requirements Elicitation: Overview

- Challenging activity
- People with **different backgrounds** must collaborate
  - Client and end users with **application (problem) domain knowledge**
  - Developer with **solution domain knowledge** (design knowledge, implementation knowledge)
- Difficulties
  - Identifying an appropriate system
  - Communicating about the domain and the system accurately

Types of Requirements Elicitation

- Greenfield Engineering
  - **Development from scratch**, no prior system exists
  - Requirements extracted from end users and 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 services of existing system in **new environment**
  - Triggered by technology enabler or new market needs

Problem Statement

- Developed by the client as a description of the **problem addressed by the system**
- Synonym: Statement of work
- A 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 (milestones)
  - A set of acceptance criteria (criteria for system tests)

Current Situation

- Describes the problem to be solved
- Describes the motivation (business requirement)
  - A change in the application domain or in the solution domain
- Change in the application domain
  - A new function (business process) is introduced
- Change in the solution domain
  - A new solution (technology enabler) has appeared

Bankomat: The Problem

- Business need
  - Providing standard services (withdrawals, transfers, etc.) to bank clients is labor-intensive and expensive
  - Due to market pressure, bank fees have been decreasing
- Customer request
  - Customers want to use basic bank services outside the normal business hours
- Technological advance
  - Computers and networks enable development of automatic service machines

Bankomat: Objectives

- Provide software for operating a banking machine
  - Withdraw money in two currencies (CHF / €)
  - Transfer money to domestic accounts
  - Load and unload cash cards
  - Print account statements
- Provide functionality to satisfy legal documentation regulations

Types of Requirements

Functional Requirements

- Functionality
  - What is the software supposed to do?
- External interfaces
  - Interaction with people, hardware, other software

Nonfunctional Requirements

- Performance
  - Speed, availability, response time, recovery time
- Attributes (quality requirements)
  - Portability, correctness, maintainability, security
- Design constraints
  - Required standards, operating environment, etc.

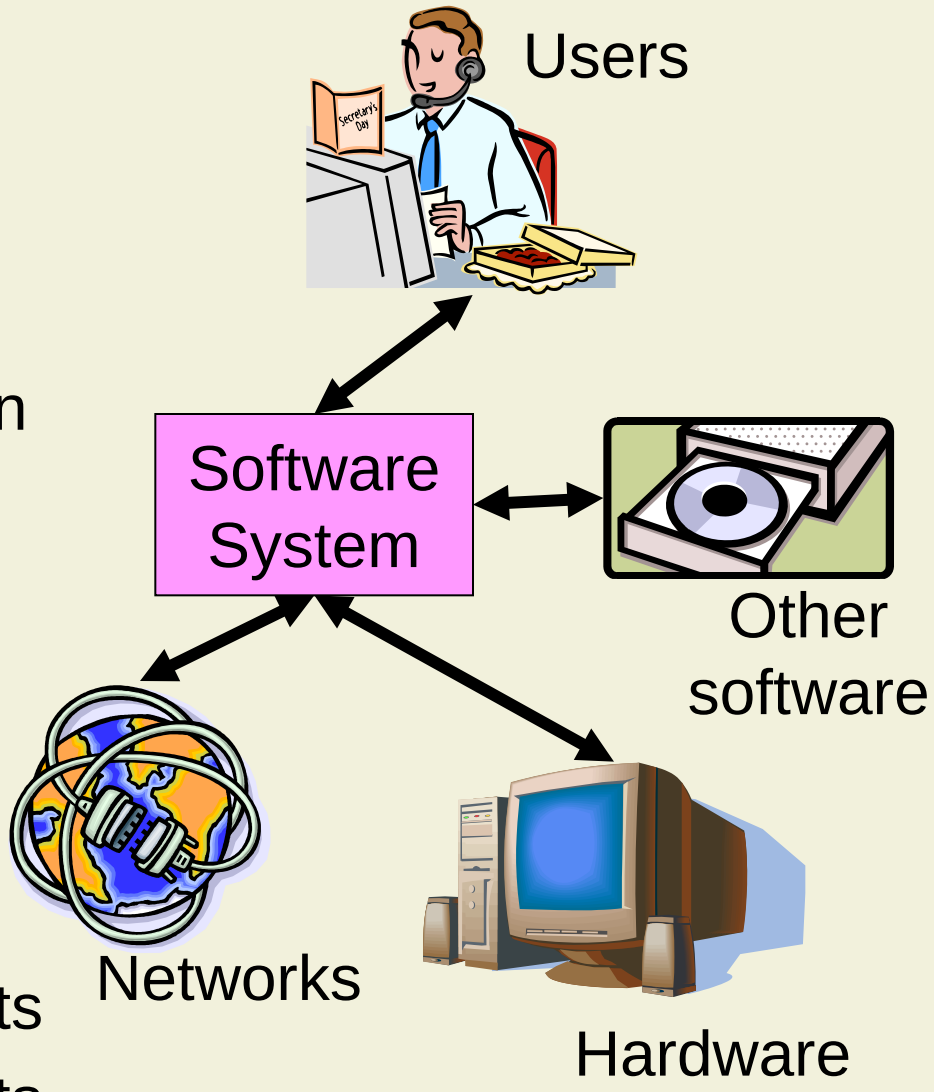
Functionality

- Includes
  - Relationship of **outputs** to **inputs**
  - Response to **abnormal situations**
  - **Exact sequence** of operations
  - **Validity checks** on the inputs
  - Effect of **parameters**

- Phrased as an action or a verb
  - Withdraw money
  - Load cash card

External Interfaces

- Detailed description of all inputs and outputs
  - Description of purpose
  - Source of input, destination of output
  - Valid range, accuracy, tolerance
  - Units of measure
  - Relationships to other inputs/outputs
  - Screen and window formats
  - Data and command formats



Performance

- Static numerical requirements
  - Number of terminals supported
  - Number of simultaneous users supported
  - Amount of information handled

- Dynamic numerical requirements
  - Number of transactions processed within certain time periods (average and peak workload)
  - Example: 95% of the transactions shall be processed in less than 1 second

Constraints (Pseudo Requirements)

- Standard compliance
  - Report format, audit tracing, etc.
- Implementation requirements
  - Tools, programming languages, etc.
  - Development technology and methodology should not be constrained by the client. Fight for it!
- Operations requirements
  - Administration and management of the system
- Legal requirements
  - Licensing, regulation, certification

Nonfunctional Requirements: Bankomat

- Usability
  - User interaction shall be done via a touch-screen
  - Text shall appear in letters at least 1cm high
- Security
  - System shall under no circumstances leak PIN numbers or account information to unauthorized users
- Performance
  - Each individual transaction shall take less than 10s
- Operations requirements
  - System updates shall be possible remotely

Quality Criteria for Requirements

Correctness
Requirements represent the client's view

Completeness
All possible scenarios are described, including exceptional behavior

Consistency
Requirements do not contradict each other

Clarity (Un-ambiguity)
Requirements can be interpreted in only one way

Quality Criteria for Requirements (cont'd)

Realism
Requirements can be implemented and delivered

Verifiability
Repeatable tests can be designed to show that the system fulfills the requirements

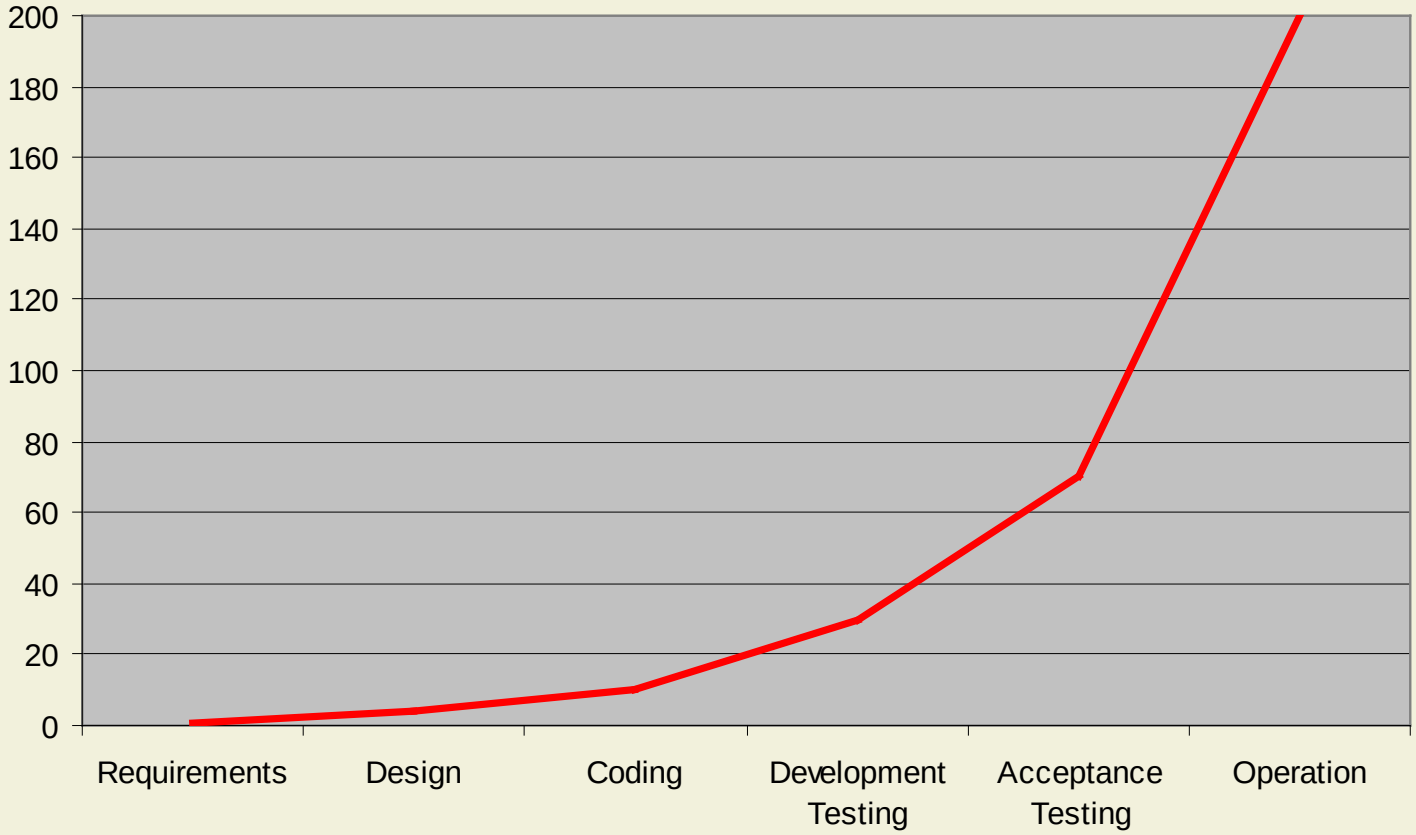
Traceability
Each feature can be traced to a set of functional requirements

Quality Criteria: Examples

- "*System shall be usable by elderly people*"
  - Not verifiable, unclear
  - Solution: "*Text shall appear in letters at least 1cm high*"
- "*The product shall be error-free*"
  - Not verifiable (in practice), not realistic
  - Solution: Specify test criteria
- "*The system shall provide real-time response*"
  - Unclear
  - Solution: "*The system shall respond in less than 2s*"

Relative Cost to Fix an Error

- The sooner a defect is found, the cheaper it is to fix



[Boehm 1981]

Requirements Validation

- A quality assurance step, usually after requirements elicitation or analysis
- **Reviews** by clients and developers
  - Check all quality criteria
  - Future validations (testing)
- **Prototyping**
  - Throw-away or evolutionary prototypes
  - Study feasibility
  - Give clients an impression of the future system
  - Typical example: user interfaces

2. Requirements Elicitation

2.1 Requirements

2.2 Documenting Functional Requirements

2.3 Requirements Elicitation Activities

2.4 Requirements Documentation

Scenarios and Use Cases

- Document the behavior of the system from the **users' point of view**
- Can be **understood by customer and users**

Scenario

- Describes **common cases**
- Focus on **understandability**

Use Case

- Generalizes scenarios to describe **all possible cases**
- Focus on **completeness**

- A scenario is an instance of a use case

Scenarios

- Definition:
  A narrative description of what people do and experience as they try to make use of computer systems and applications

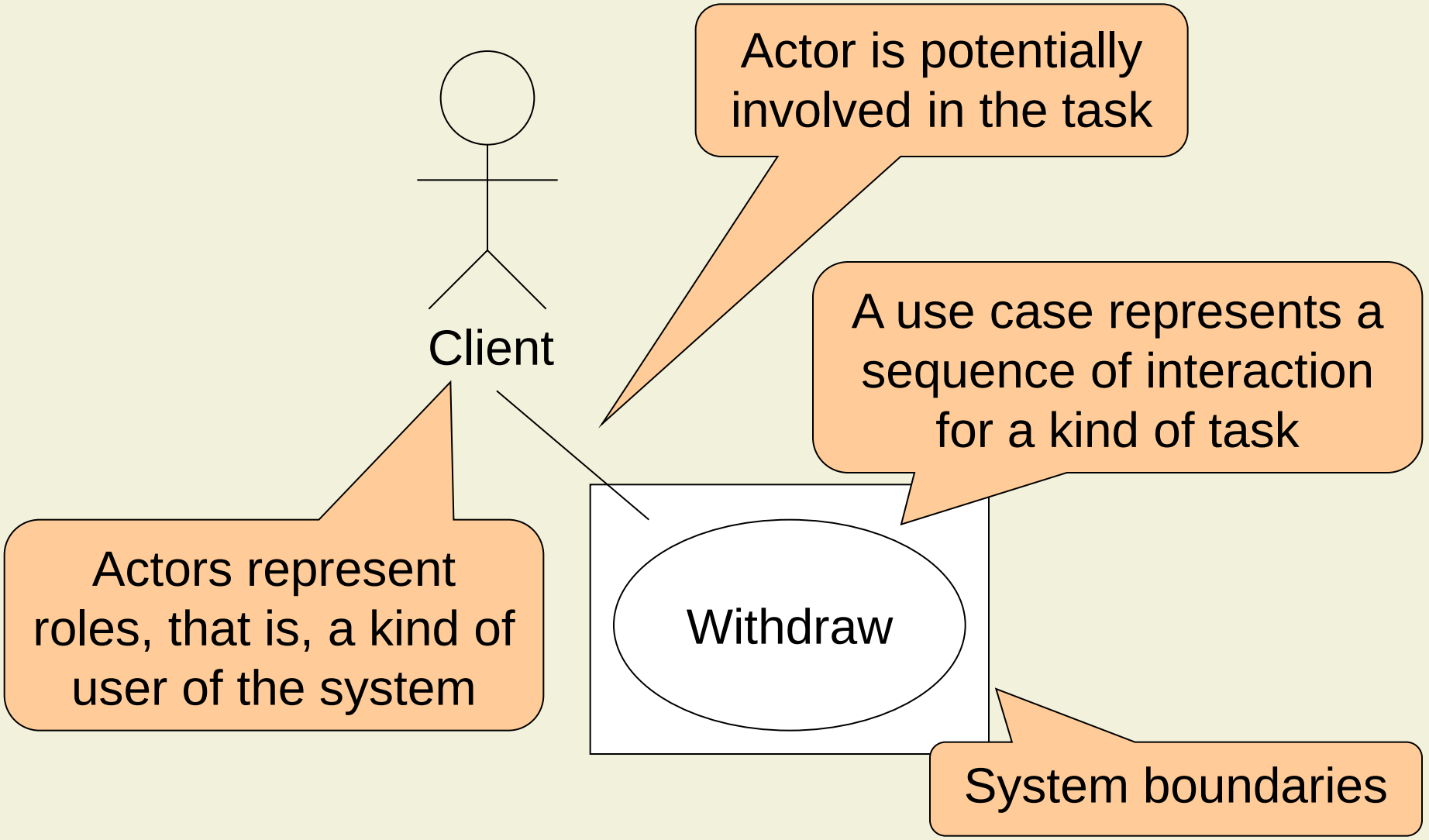
  [M. Carroll, 1995]

- Different Applications during the software lifecycle
  - Requirements Elicitation
  - Client Acceptance Test
  - System Deployment

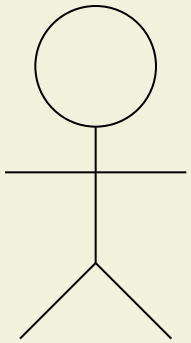
Scenario Example: Bankomat

- Scenario
  - Bob uses a Bankomat
  - He enters his card and PIN into the Bankomat
  - He requests withdrawal of CHF 400
  - Bob receives a printed receipt, takes out his bank card and the money and leaves
- Observations
  - Describes a single instance of using the system
  - Does not describe all possible ways the system can be used

UML Use Case Diagrams

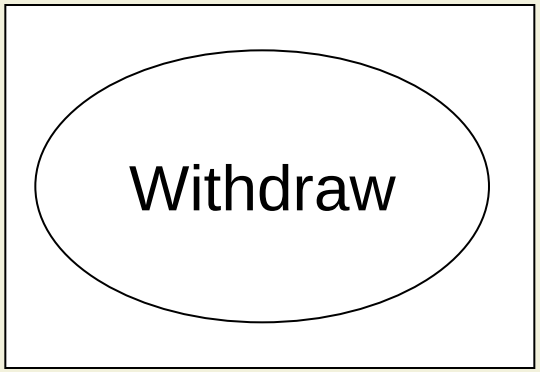


Actors



- An actor models an **external entity** which communicates with the system
  - Kind of user
  - External system
  - Physical environment
- An actor has a unique name and an optional description
  - Client: A person in the train
  - GPS satellite: An external system that provides the system with GPS coordinates

Use Case



- A use case represents a **kind of task** provided by the system as an event flow
- A use case consists of
  - Unique name
  - Participating actors
  - Entry conditions
  - Flow of events
  - Exit conditions
  - Special requirements

Use Case Example: Withdraw

- Initiating actor: Client

- Entry condition
  - Client has opened a bank account with the bank and
  - Client has received a bank card and PIN

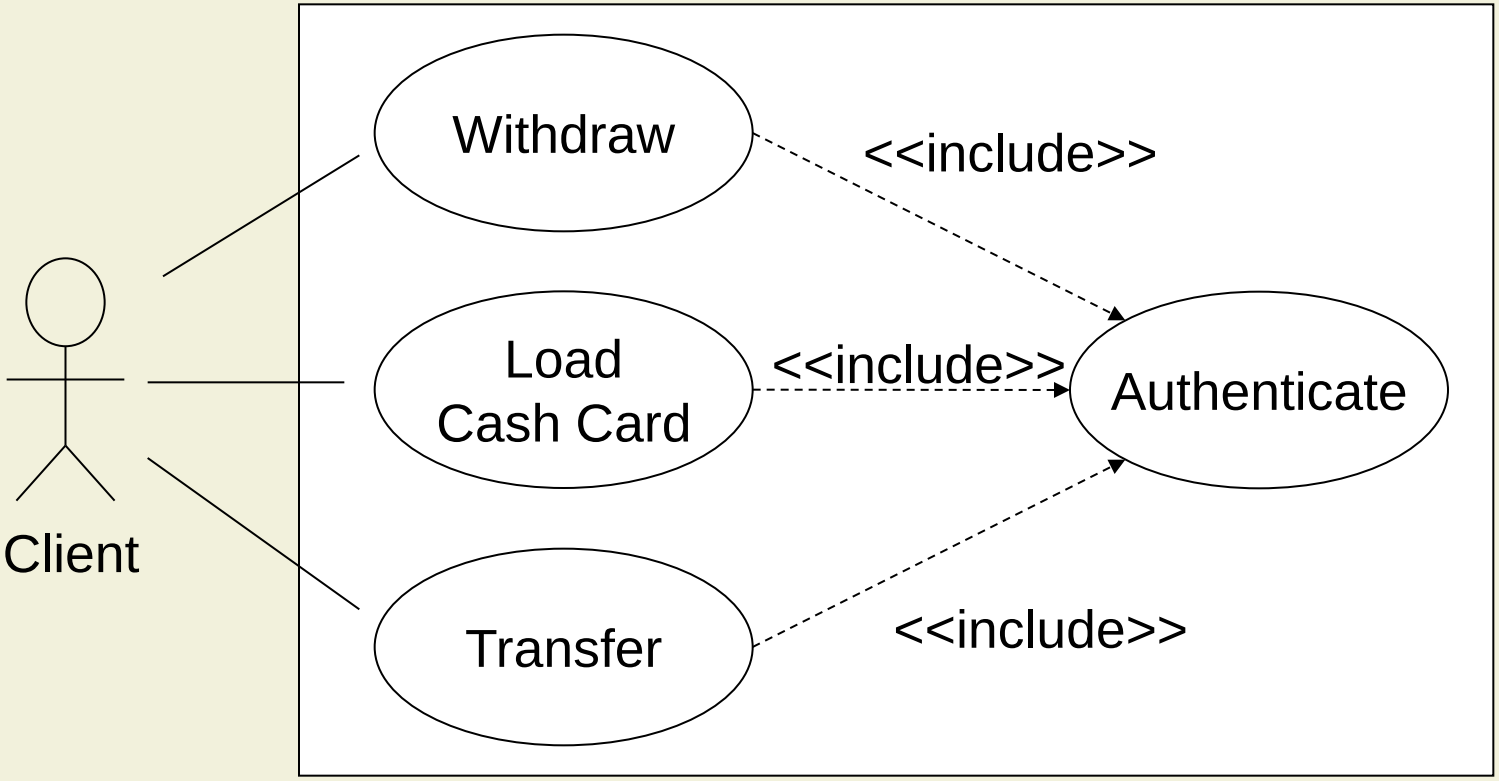
- Exit condition
  - Client has the requested cash or
  - Client receives an explanation from the Bankomat about why the cash could not be dispensed

Use Case Example: Withdraw Event Flow

Actor steps	System Steps
1. Authenticate	
	2. Bankomat displays options
3. Client selects “Withdraw CHF”	
	4. Bankomat queries amount
5. Client enters amount	
	6. Bankomat returns bank card
	7. Bankomat outputs specified amount in CHF

Anything missing?
Exceptional cases,
Details of authentication

Reusing Use Cases

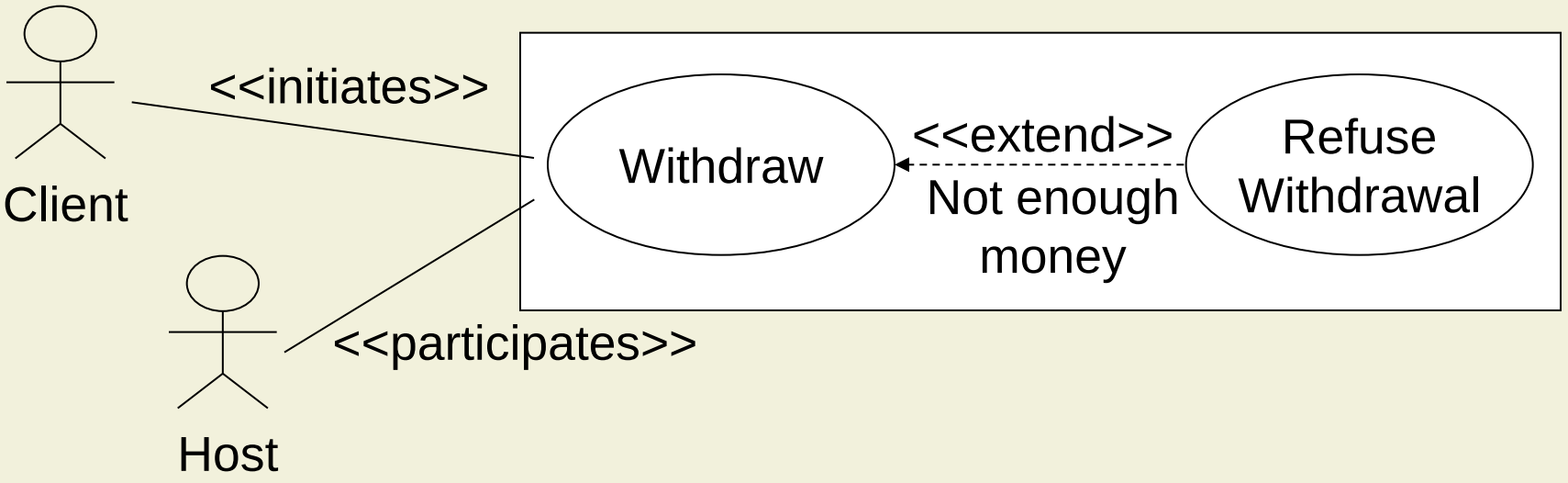


- <<include>> stereotype to include use cases
- Details in textual description

Reusing Use Cases: Discussion

- **Pros**
  - Convenient (no duplicate information in detailed description)
  - Shorter descriptions
  - Common functionality may lead to reusable components
  - Enables integration of existing components
- **Cons**
  - May lead to functional decomposition rather than object-oriented model
  - Requires more UML skills
- Criterion for decomposition:
  Size of planning unit (40-80 person hours)

Separating Variant Behavior



- <<extend>> stereotype to provide special case
- Normal case specifies point at which the behavior may diverge (**extension point**)
- Extending case specifies condition under which the special case applies (as **entry condition**)

Withdraw Event Flow Revisited

Actor steps	System Steps
1. Authenticate **(use case Authenticate)**	
	2. Bankomat displays options
3. Client selects “Withdraw CHF”	
	4. Bankomat queries amount
5. Client enters amount	
	6. Bankomat returns bank card
	7. Bankomat outputs specified amount in CHF

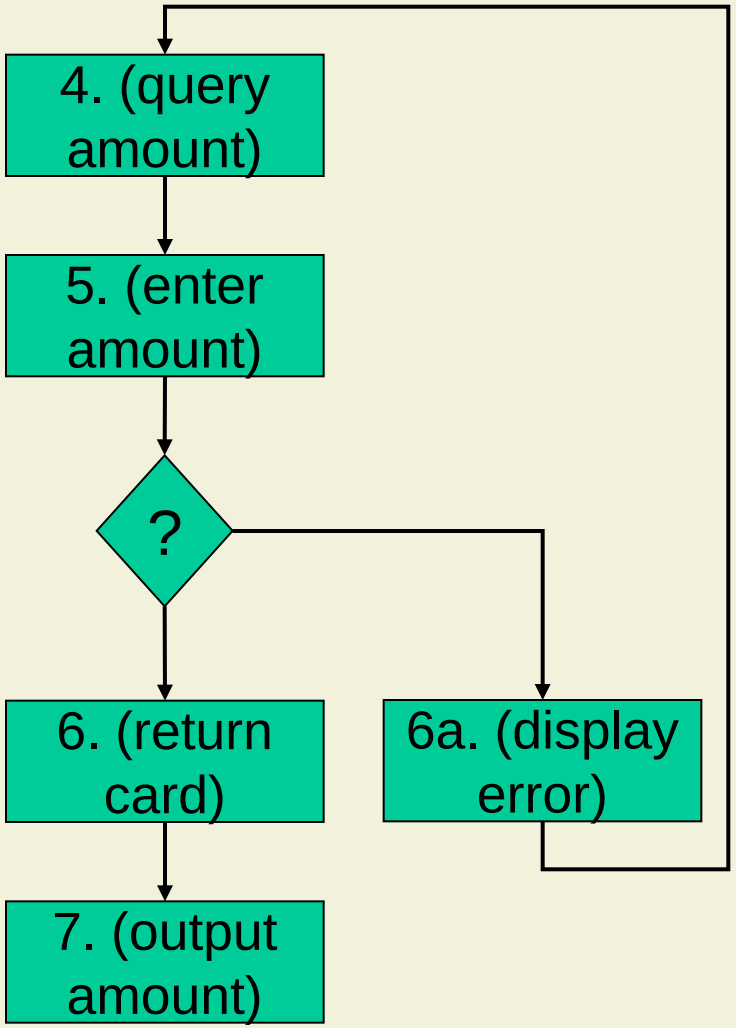
Listed as extension point

Use Case Refuse Withdrawal

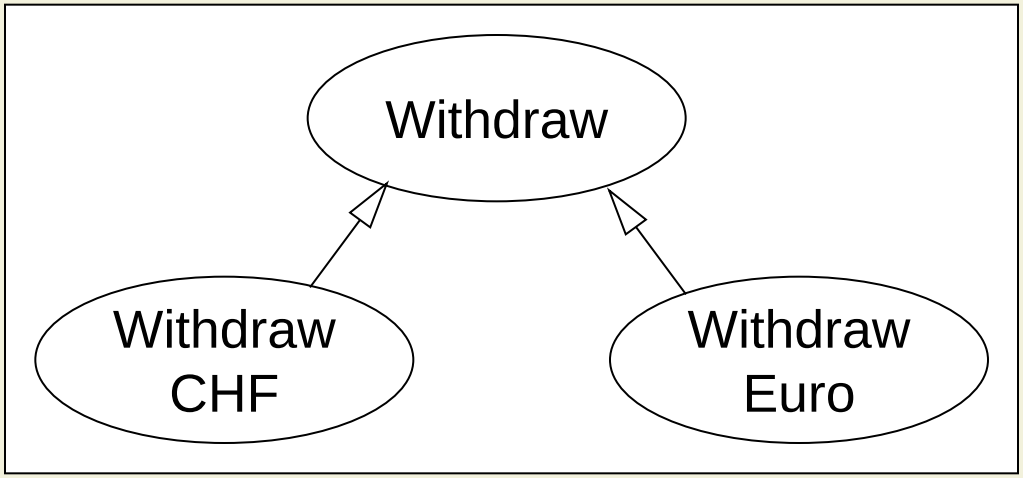
Entry Condition:
Entered amount higher than money in account

System Steps:

6a. Bankomat displays error message; rejoin before 4.



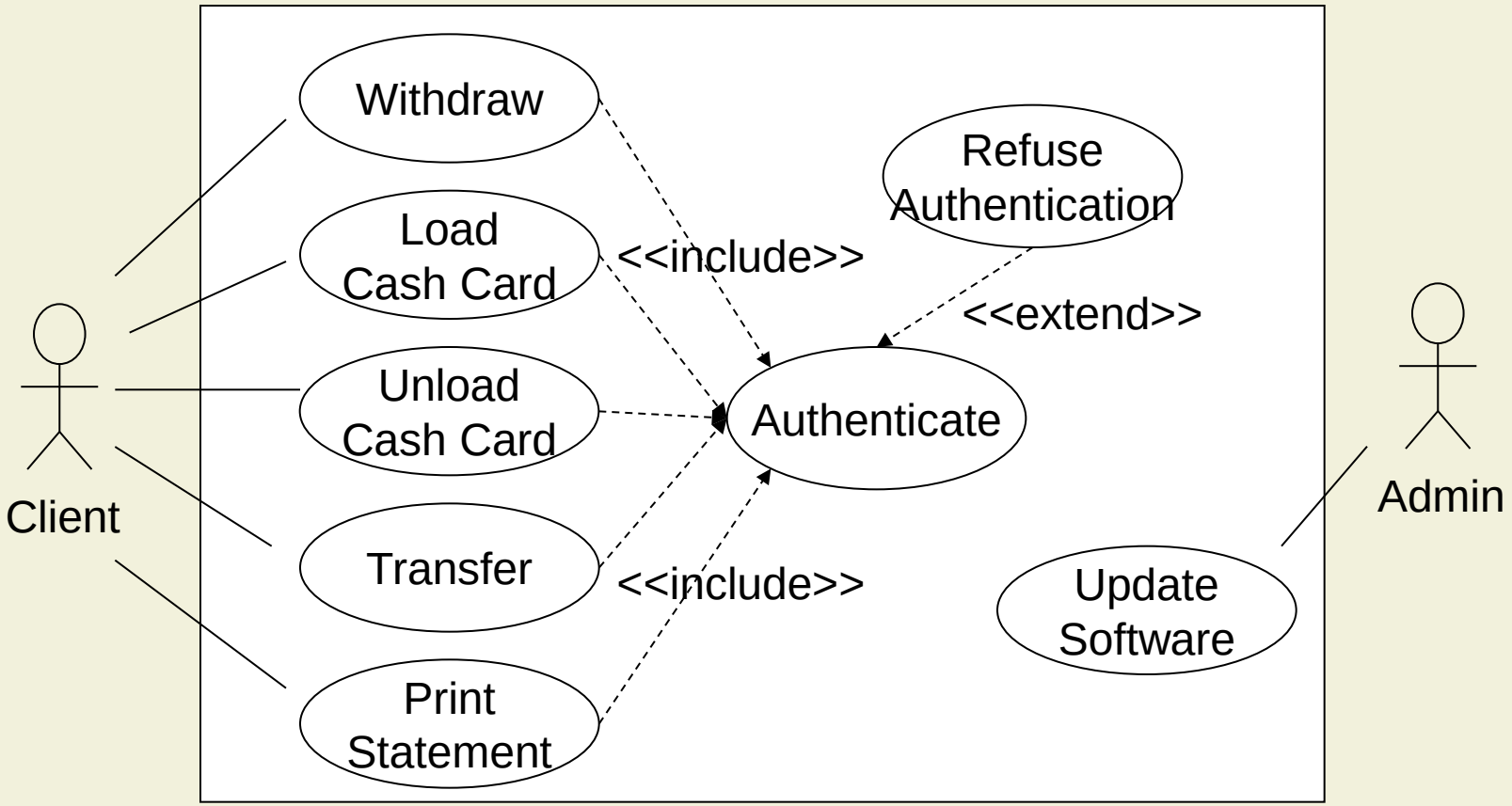
Generalization and Specialization



- Factor out **common** (but not identical) **behavior**
- Child use cases
  - **Inherit** the behavior and meaning of the parent use case
  - **Add** or **override** some behavior
- Details in textual description of normal case

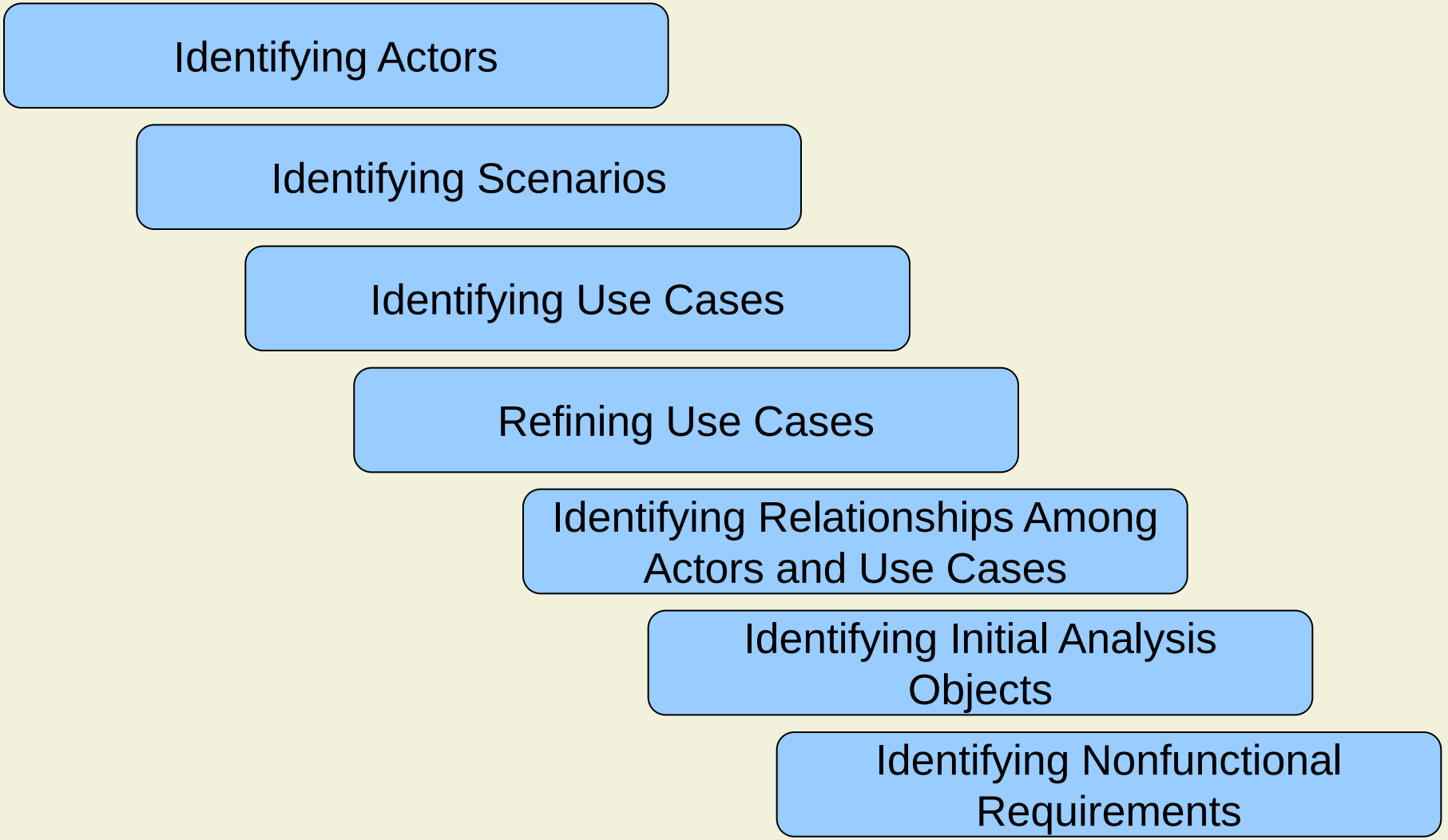
Use Case Models

- The set of all use cases specifying the **complete functionality** of the **system** and its **environment**



2. Requirements Elicitation

Requirements Elicitation Activities



Identifying Actors

- Actors represent **roles**
  - One person can have several roles
  - Many persons can have the same role
  - In companies, roles usually exist before system is built
- **Questions** to ask
  - Which user groups are supported by the system?
  - Which user groups execute the system's main functions?
  - Which user groups perform secondary functions (maintenance, administration)?
  - With what external hardware and software will the system interact?

Actors vs. Objects

- During initial stages of actor identification, it is **difficult to distinguish actors from objects**

- Example: database system can be
  - An actor (external software) or
  - An object (part of the system)

- Problem is solved when **system boundaries** are defined
  - Actors are outside
  - Objects are inside

Identifying Scenarios: Questions to Ask

- What are the tasks 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?

Example: Bankomat

- What needs to be done to withdraw money?
- Who is involved in a withdrawal?
- What does the system do if there is not enough
  - Money in the account?
  - Cash in the Bankomat?
- What information does the client access?
- Can clients perform several tasks in one session?

Sources of Information

Client

Speak to the end user, not just to the client

Users

- User manuals
- Procedure manuals
- Company standards
- etc.

Elicitation

Existing documentation

Insist on task observation

Task observation

Dialectic Approach

- Apply **evolutionary, incremental** engineering
  - 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

- Client understands problem domain, not the solution domain.
  - Write scenarios using **application domain terms**
  - Example: “Client” instead of “Account ID”

Types of Scenarios

Visionary scenario

- Used to describe a future system
- Usually used in greenfield engineering and reengineering projects
- Can often not be done by the user or developer alone

As-is scenario

- Used in describing a current situation
- Usually used in re-engineering projects
- The user describes the system

Evaluation scenario

- User tasks against which the system is to be evaluated

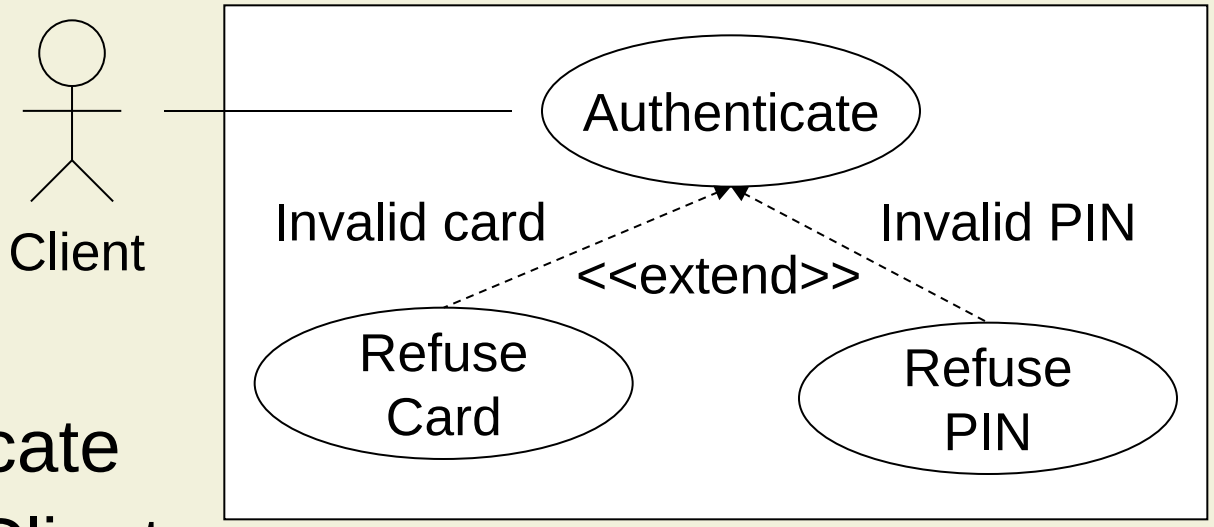
Training scenario

- Step by step instructions that guide a novice user through a system

Identifying Use Cases

- Scenarios are **generalized** to high-level use cases
- Name
  - A verb describing what the actor want to accomplish
- Initiating actor
  - Helps to clarify roles
  - Helps identifying previously overlooked actors
- High-level description
  - Entry and exit conditions (identify missing cases)
  - Event flow (define system boundary)
  - Quality requirements (elicit nonfunctional requirements)

Another Use Case Example: Authenticate



- Name: Authenticate
- Initiating actor: Client
- Entry condition
  - Client has opened a bank account with the bank and
  - Client has received an bank card and PIN
- Exit condition
  - Client is authenticated

Authenticate Event Flow

Actor steps	System Steps
1. Client inputs her card into the Bankomat	
	2. Bankomat requests the input of a four-digit PIN
3. Client types in PIN	

Listed as extension point

- Triggers for extending use cases (error conditions) are specified in extending use cases (as entry conditions)

Use Case Refuse Card

- Name: Refuse authentication
- Entry condition
  - **Client used invalid card**
- Exit condition
  - Client receives an explanation from the Bankomat about why she was not authenticated

Actor steps	**System Steps**
	2a. Bankomat outputs the card, displays a message, and stops the interaction

Guidelines for Use Cases

- Name
  - Use a **verb phrase** to name the use case
  - The name indicates what the user is trying to accomplish
  - Examples: “Withdraw”, “Authenticate”, “Load Cash Card”

- Length
  - A use case should not exceed two A4 pages
  - If longer, use <<include>> relationships
  - A use case describes a complete set of interactions

Guidelines for Use Cases (cont'd)

- Flow of events
  - Active voice is used
  - Steps start either with "The Actor …" or "The System …"
  - The causal relationship between steps is clear
  - All flow of events are described (not only main flow)
  - The boundaries of the system are clear
  - Important terms are defined in the glossary

A Poor Use Case

- Name: Cash Card
- Initiating actor: Client
- Flow of events
  1. The client enters his card and PIN
  2. Cash card is loaded with specified amount

Bad name:
What is the user trying to accomplish?

Causality:
- What causes the card to be loaded?
- Who specifies the amount?

Passive voice:
Who loads the card?

Incomplete transaction:
What happens after the card is loaded?

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

Refining Use Cases

- Many use cases are rewritten several times
- Focus: **completeness** and **correctness**
- Activities during refinement
  - Add details to use cases
  - Specify low-level sequences of interactions
  - Specify access rights
    (which actor can invoke which use case)
  - Identify missing exceptions and specify handling
  - Factor out common functionality

Relationships Among Actors and Use Cases

- Communication relationships between actors and use cases: <<initiate>> vs. <<participate>>

- Extend relationships
  - Make common case simple
  - Used for exceptional, optional, or seldom-occurring behavior
- Include relationships
  - Eliminate redundancies
  - Used for behavior shared by at least two use cases

Identifying Nonfunctional Requirements

- **Nonfunctional** requirements are defined **together with functional** requirements because of dependencies
  - Example: Support for novice users requires help functionality
- Elicitation is typically done with **check lists**
- Resulting set of nonfunctional requirements typically contains **conflicts**
  - Real-time requirement needs C or Assembler implementation
  - Supportability requires OO-implementation

2. Requirements Elicitation

2.1 Requirements

2.2 Documenting Functional Requirements

2.3 Requirements Elicitation Activities

2.4 Requirements Documentation

Requirements Analysis Document

1. Introduction
   1. Purpose and scope of the System
   2. Objectives and success criteria of the project
   3. Definitions, acronyms, references, overview
2. Current System
3. Proposed System
   1. Overview
   2. Functional requirements
   3. Nonfunctional requirements
   4. System models
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: Checklist

3.3.1 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 sort of training will be required for each type of user?
- Is it particularly important that the system be easy to learn?
- Is it important that users be protected from making errors?
- What sort of input/output devices for the human interface are available, and what are their characteristics?

3.3.2 Documentation

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

Nonfunctional Requirements: Checklist (cont'd)

3.3.3 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?

3.3.4 Performance characteristics

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

3.3.5 Error handling and extreme conditions

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

Nonfunctional Requirements: Checklist (cont'd)

3.3.6 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?

3.3.7 Quality Issues

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

Nonfunctional Requirements: Checklist (cont'd)

3.3.8 System Modifications

- What parts of the system are likely candidates for later modification?
- What sorts of modifications are expected?

3.3.9 Physical Environment

- Where will the target equipment operate?
- Will the target equipment be in one or several locations?
- Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, ...)?

Nonfunctional Requirements: Checklist (cont'd)

3.3.10 Security Issues

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

3.3.11 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?

Prioritizing Requirements

- High priority ("Core requirements")
  - Must be addressed during **analysis**, **design**, and **implementation**
  - A high-priority feature must be demonstrated successfully during **client acceptance**
- Medium priority ("Optional requirements")
  - Must be addressed during **analysis** and **design**
  - Usually implemented and demonstrated in the **second iteration** of the system development
- Low priority ("Fancy requirements", "nice to have")
  - Must be addressed during **analysis**

Project Agreement

- **Acceptance** of (parts of) the analysis model (as documented by the requirements analysis document) **by the client** (client sign-off)
- The client and the developers **agree** about the **functions and features** that the system will have, plus:
  - A list of prioritized requirements
  - A revision process
  - A criteria that will be used to accept or reject the system
  - A schedule and a budget

Summary

- Scenarios: Good way to **establish communication** with client
- Use cases: Abstraction of scenarios
- Pure functional decomposition is bad
  - Leads to un-maintainable code
- Pure object identification is bad
  - May lead to wrong objects, attributes, and methods
- Use cases **bridge the gap** between functional requirements and objects

From Scenarios to Use Cases

- First step: name the use case
  - Example: Withdraw
- Second step: find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Client”)
  - Participating Actors: Client, host system
- Third step: concentrate on the flow of events
  - Use informal natural language