Software Architecture and Engineering

Requirements Elicitation

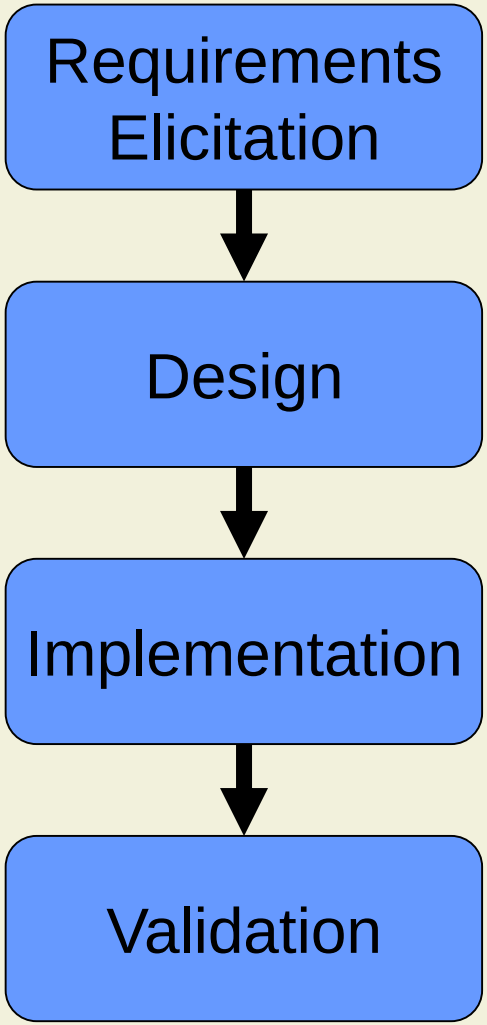
Peter Müller

Chair of Programming Methodology

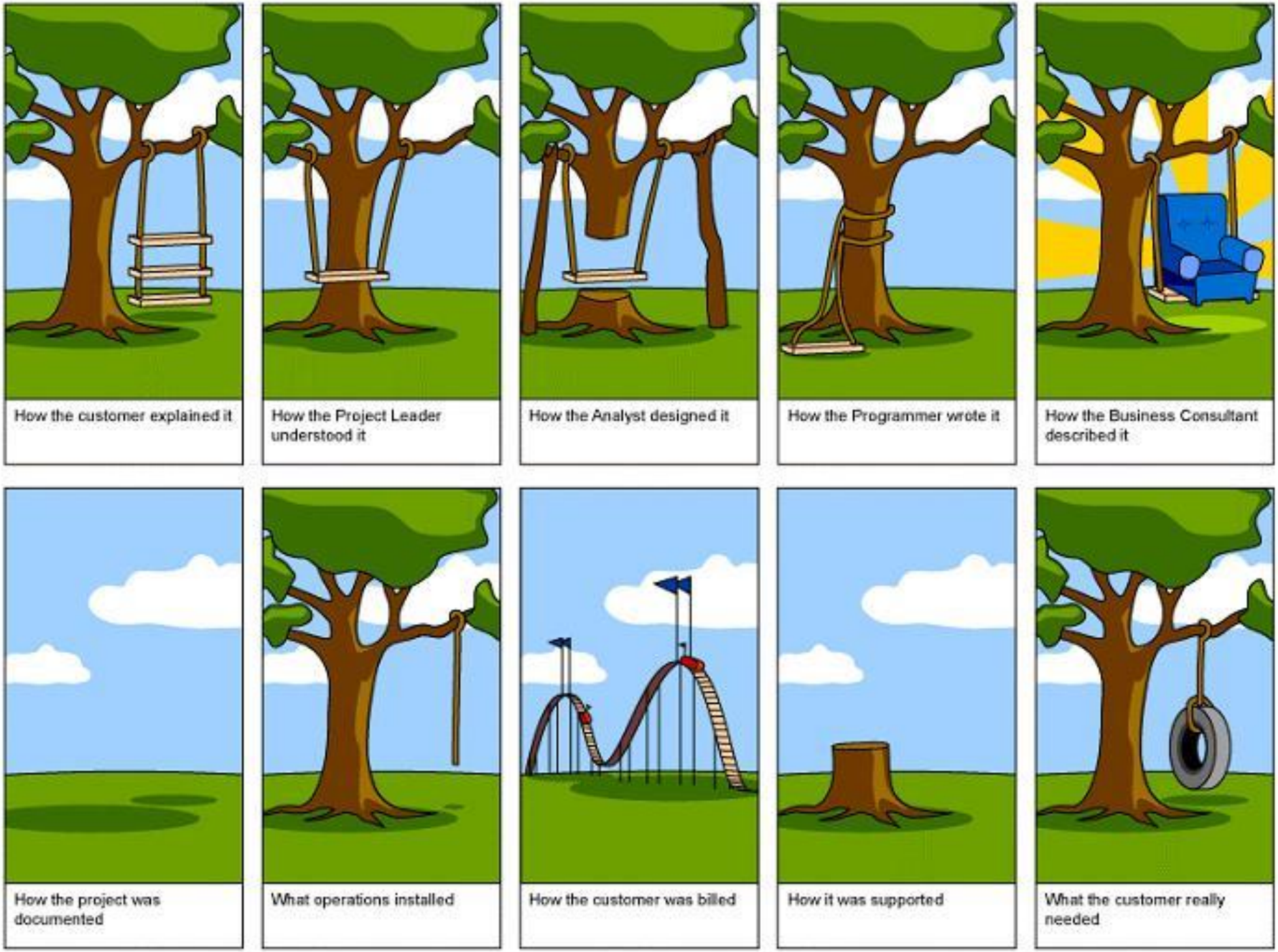
Spring Semester 2017

ETH zürich

Main Activities of Software Development

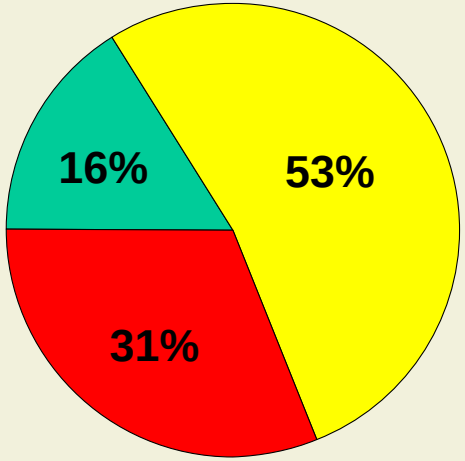


These activities may overlap and are typically executed iteratively


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

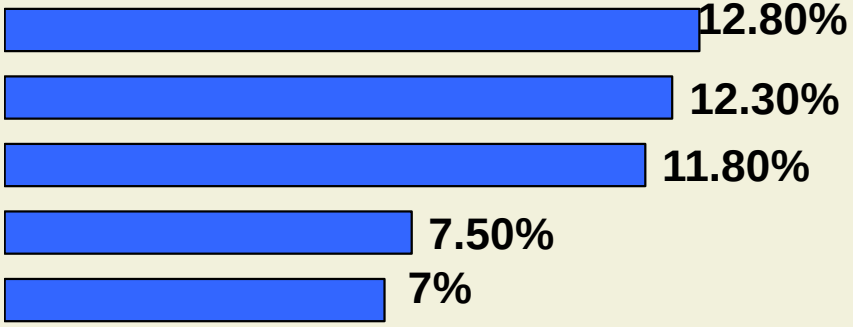
Software – a Poor Track Record

- 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

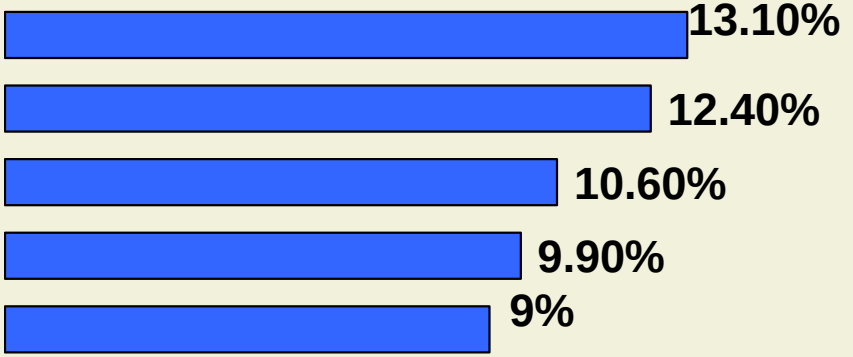


Why IT-Projects Fail

- 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

Types of Requirements

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

Functional Requirements

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

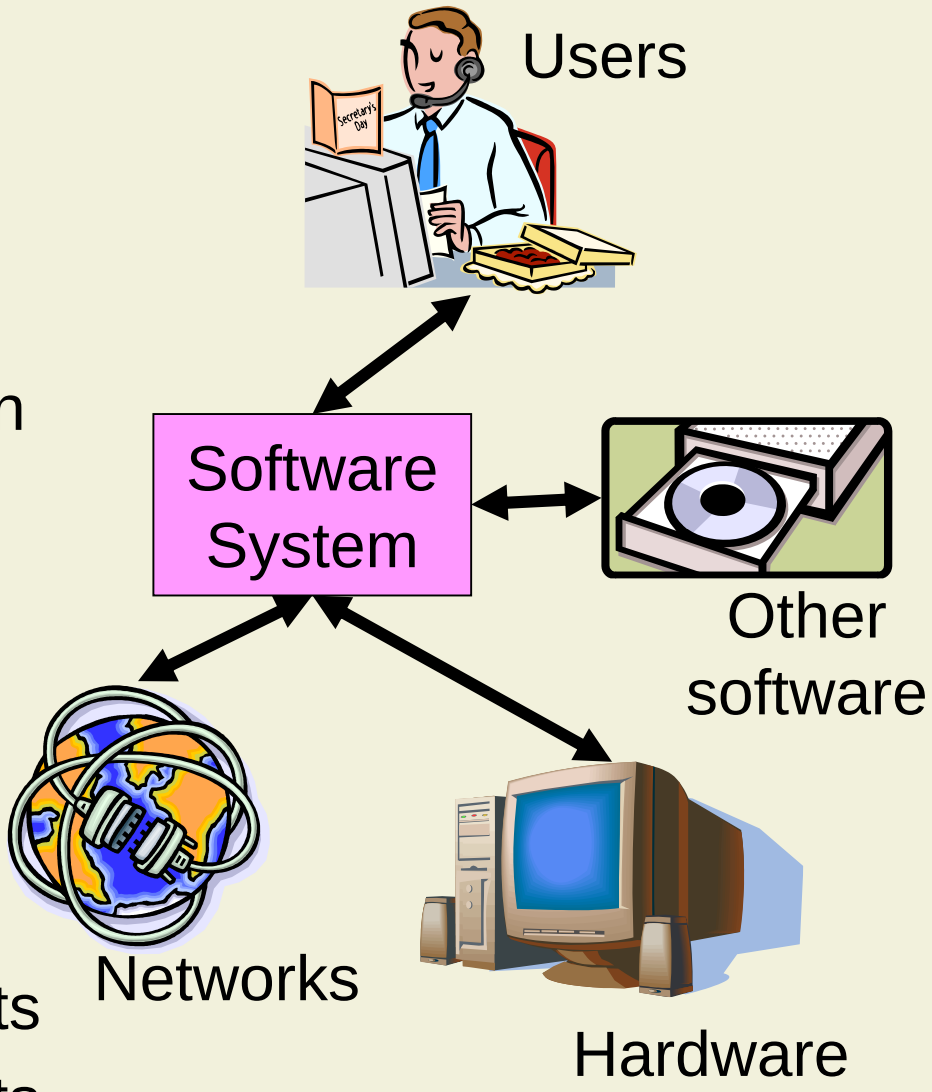
Nonfunctional Requirements

Functionality

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

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

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

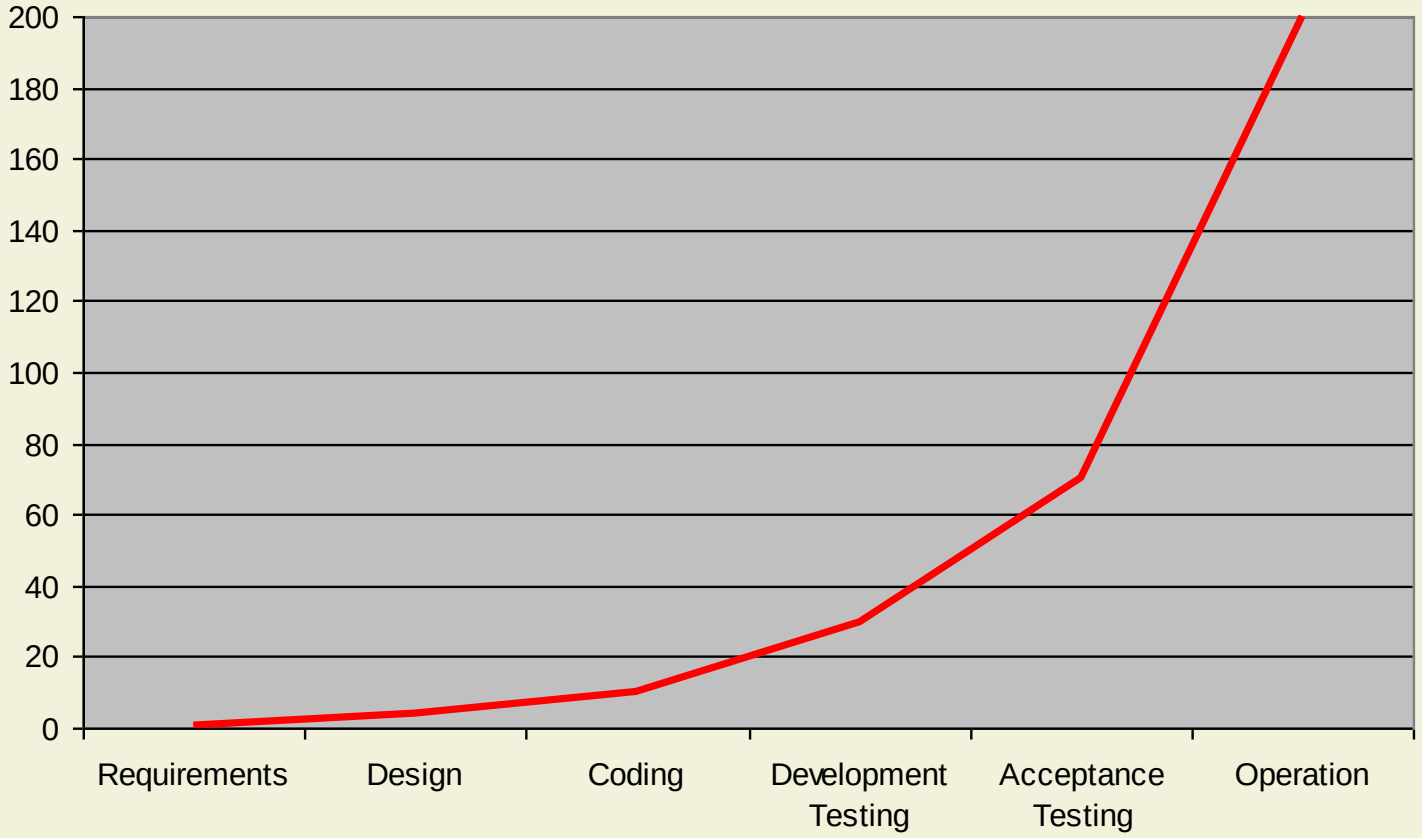
Traceability
Each feature can be traced to a set of functional requirements

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 Activities

Requirements Elicitation Activities

- Identifying Actors
- Identifying Scenarios
- Identifying Use Cases
- Identifying Nonfunctional Requirements

Identifying Actors

- Actors represent roles
  - Kind of user
  - External system
  - Physical environment
- 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?

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

When Alice wants to borrow a book, she takes it to the checkout station. There she first scans her personal library card. Then she scans the barcode label of the book. If she has no borrowed books that are overdue and the book is not reserved for another person, the systems registers the book as being borrowed by her and turns off the electronic safety device of that book. Several books can be checked out together. The checkout procedure is terminated by pressing a ‘Finished’ key. The system produces a loan slip for the books that have been borrowed.

[Adapted from Glinz 2000]

Identifying Scenarios: Questions to Ask

- What are the tasks the actor wants the system to perform?
- What information does the actor access?
- Which external changes does the actor need to inform the system about?
- Which events does the system need to inform the actor about?

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

Task observation

Use Cases

- A list of steps describing the interaction between an actor and the system, to achieve a goal

- A use case consists of
  - Unique name
  - Initiating and participating actors
  - Flow of events
  - Entry conditions
  - Exit conditions
  - Exceptions
  - Special requirements

Use Case Example: Event Flow

Actor steps	System Steps
1. Scans library card	2. validates the card; returns the card; displays user data; displays ‘Select function’ dialog
3. selects ‘Borrow’ function	4. displays ‘Borrow’ dialog
5. scans label of book to be borrowed	6. identifies book; records book as borrowed, unlocks safety label; displays book data
7. presses ‘Finish’ key	8. prints loan slip; displays ‘Finished’ message

Also specify alternative flows and exceptional 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 suggests C or Assembler implementation
  - Maintainability suggests OO-implementation