# Software Engineering

## Requirements Engineering II

Software & Systems Engineering | Prof. Dr. Andreas Vogelsang | 18.10.2023

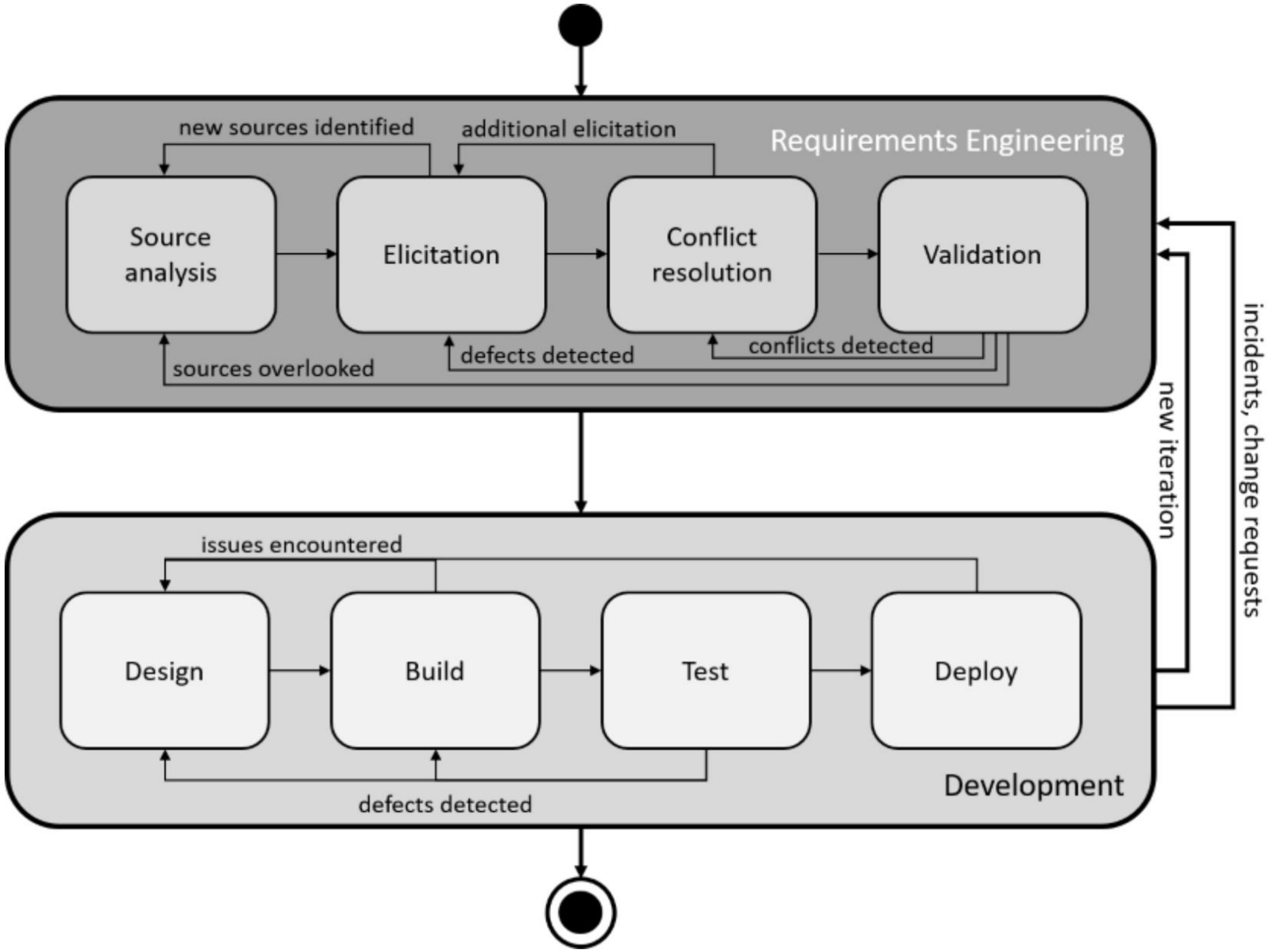
@andivogelsang
vogelsang@cs.uni-koeln.de

# Learning Goals for Today

- Understand how to elicit good requirements
- Understand how to write down requirements
- Understand how to validate requirements

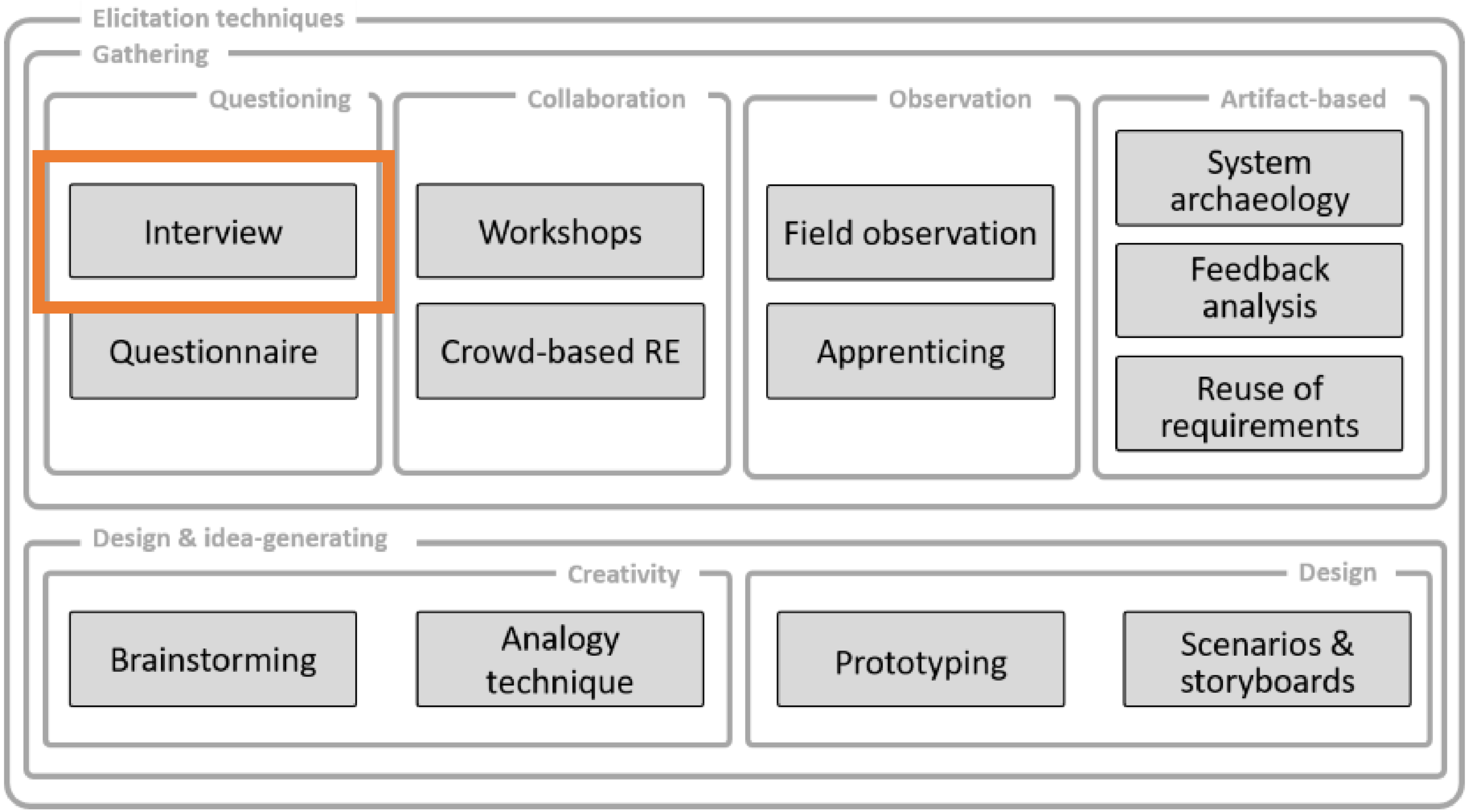
# Requirements Elaboration

# Requirements Elaboration is Not Linear



# Requirements Elicitation

# Elicitation Techniques



Universität zu Köln

# Interview

- Types of interviews
  - Open: Only topic or rough guideline is given
  - Structured: Order of the questions, exact wording, and possible answers are given
  - Semi-structured: Parts are given, parts are open
- Types of questions
  - Closed-ended: Selection from a set of predefined answers
  - Open-ended: Free response style
- *Qualitative* interviews: Open interviews with open-ended questions
  - Goal: Exploring topics in detail, discovering background and relationships
- *Quantitative* interviews: (Semi-)Structured interviews with closed-ended questions
  - Goal: Measuring quantities

Universität zu Köln

# Interview Tradeoffs

- Strengths
  - Identifying tasks, feelings, preferences of stakeholders
  - Identifying how stakeholders (want to) interact with the system
  - Identifying challenges with the current system
- Weaknesses
  - Time-consuming execution and analysis
  - Subjective, possibly inconsistent, views
  - Partial focus on (technical) trivialities
  - Organizational problems (politics/power games)
  - Results strongly depend on interviewer's skills

# Typical Mistakes in Requirements Interviews

- Question formulation
  - Asking vague questions: “What are your expectations?”, “Can you indicate the major constraints of the project?”
  - Asking technical questions: “How do you map the business goals to the system goals?”, “If the system fails, do you have a backup?”
- Question omission
  - Not asking about other stakeholders
  - Not asking follow-up or probing questions: “Why do you think that is?”, “What would need to change for you to accomplish this?”
  - Not asking about existing systems or processes
  - Not asking about priorities

Bano et al.: ”Teaching requirements elicitation interviews: an empirical study of learning from mistakes”. *REJ 2019*

# Typical Mistakes in Requirements Interviews

- Order of questions
  - Not starting an interview with building rapport (authentically)
  - Not ending an interview with a summary
  - Asking for needs before understanding the context
- Communication skills
  - Not actively listening
  - Doing an interview like an oral survey
  - Poor linguistic skills
  - Unprofessional attitude
- Teamwork and planning
  - Interrupting each other
  - No distribution of tasks or questions

Bano et al.: "Teaching requirements elicitation interviews: an empirical study of learning from mistakes". *REJ 2019*

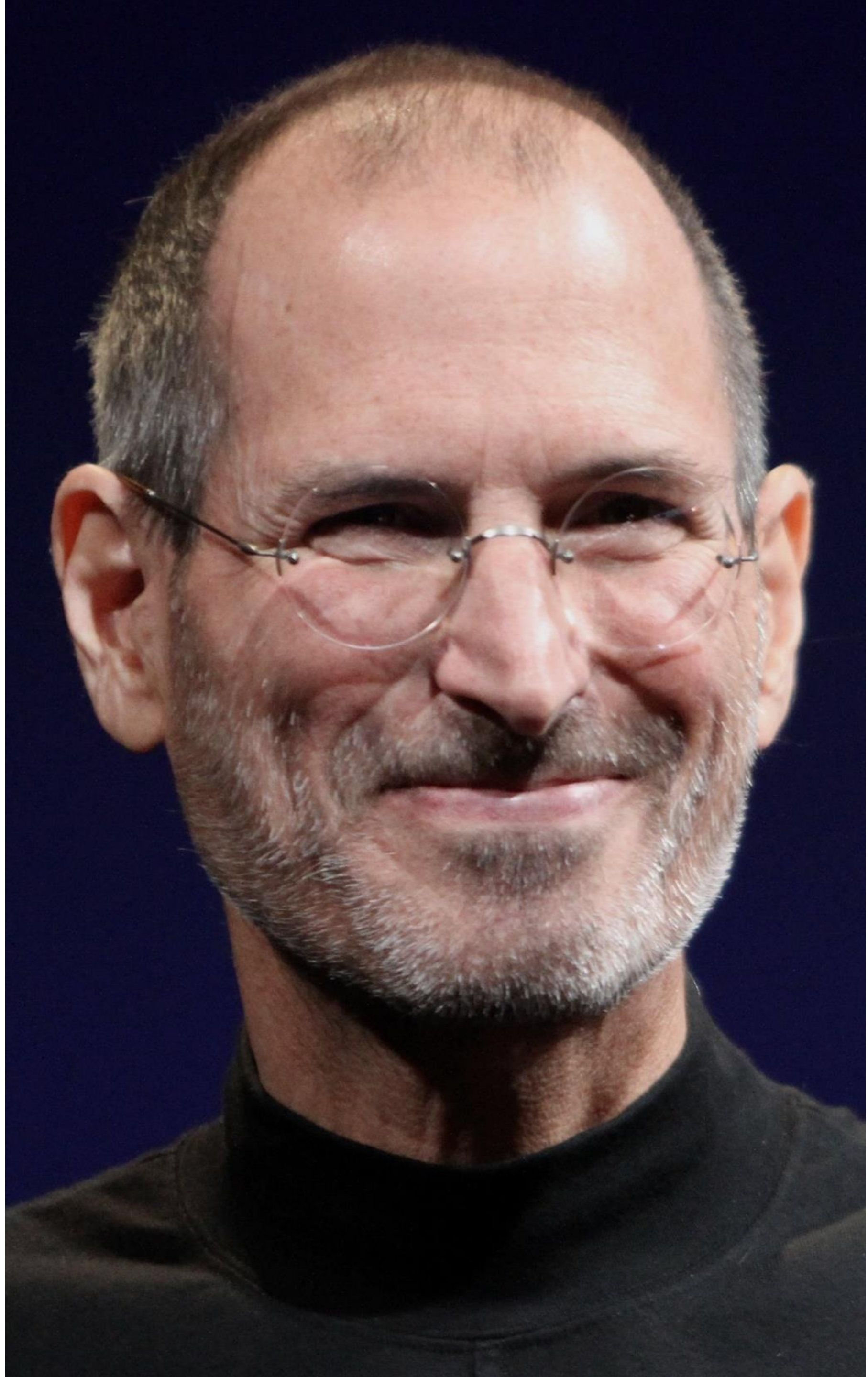
# Requirements Interview Advices

- Remember to create rapport with the customer
- Remember to identify customers' goals and success criteria
- It is important to identify goals and success criteria of different stakeholders.
- Be curious about the application domain of your customer
- Do not ask too many technical questions
- Ensure your questions are expressed in a correct manner, by rehearsing the interview
- Remember to prioritize the interview questions based on the context
- Make sure that all the relevant questions are covered, by preparing for the interview
- Make and present a summary of discussion at the end of the interview

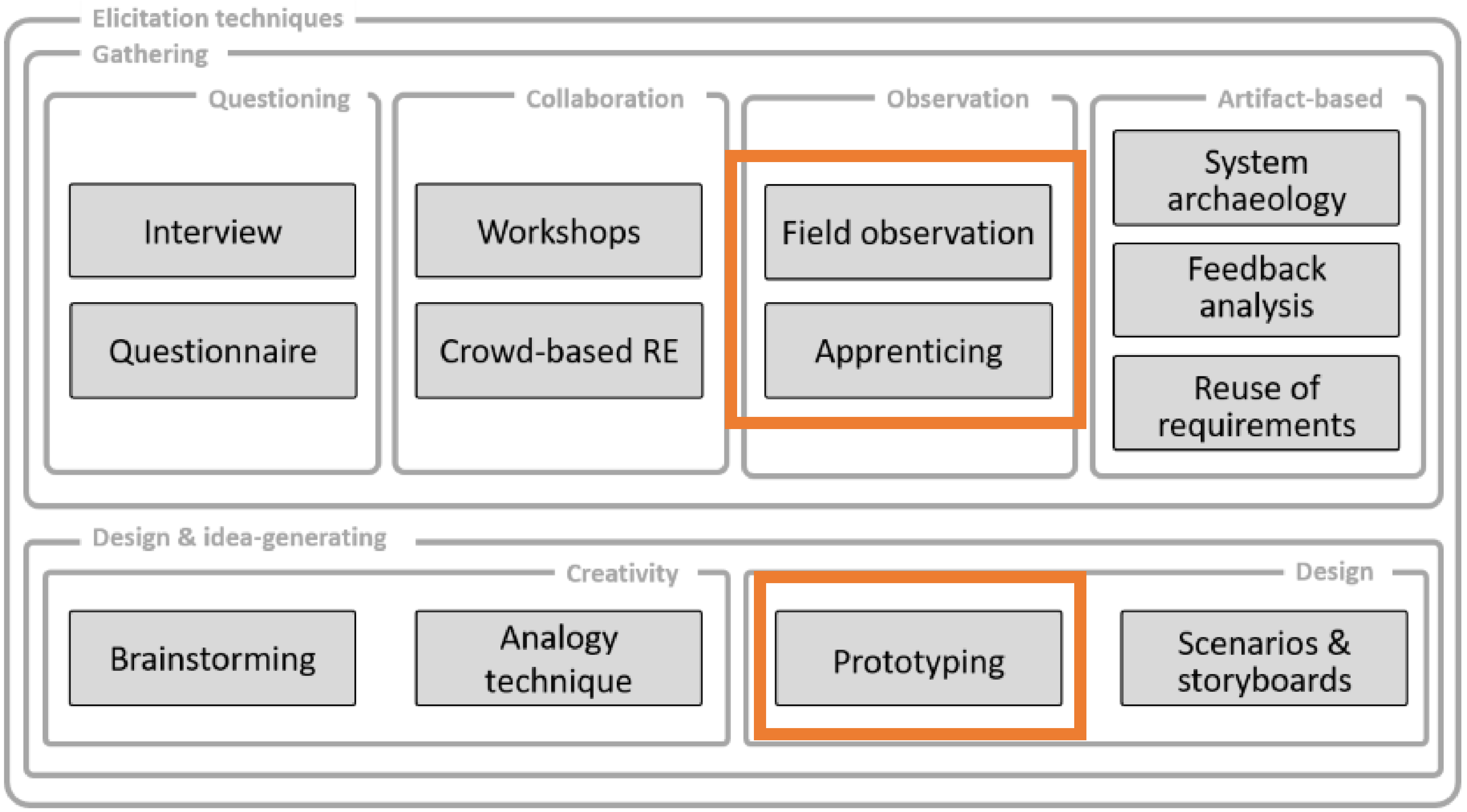
Bano et al.: "Teaching requirements elicitation interviews: an empirical study of learning from mistakes". *REJ 2019*

If I’d asked my customers what they wanted, they’d have said a faster horse.
-- Henry T. Ford

You can't just ask customers what they want and then try to give that to them. By the time you get it built, they'll want something new.
-- Steve Jobs



# Elicitation Techniques



# Synthesizing the Requirements

- Requirements are synthesized from various sources
  - Elicited from stakeholders
  - Observed in practice (maybe incl. prototypes)
  - Extracted from regulations, laws, or other documents
- And enriched with additional ideas (“Invention/Innovation”)

**Requirement (Co-)Creation**

The paper shows that about 35% of documented requirements can be fully mapped to the original customers' ideas. The rest are refinements (60%) or completely novel ideas (25%).

[Debnath et al. 21]

**In Practice**

Combinations of numerous techniques used

# Requirements Documentation

# Precision, Completeness, Consistency

## Precision

“Imprecision in the requirements specification can lead to disputes between customers and software developers. It is natural for a system developer to interpret an ambiguous requirement in a way that simplifies its implementation. Often, however, this is not what the customer wants. New requirements must be established and changes made to the system. Of course, this delays system delivery and increases costs.”

[Sommerville]

## Further desired properties

clear, easy to understand, unambiguous, correct, verifiable, prioritized, changeable, traceable

## Completeness and Consistency

“Ideally, the functional requirements specification of a system should be both complete and consistent. **Completeness** means that all services and information required by the user should be defined. **Consistency** means that requirements should not be contradictory.”

[Sommerville]

## Often not feasible in practice

mistakes, omission, implicit knowledge, many stakeholders with different backgrounds/expectations/inconsistent needs

# An Exemplary Dialog

- So, in the morning you unlock the door at the main entrance?
- Every morning?
- Even at the weekend?
- And if the plant is shut down?
- And if you are sick or in holidays?
- And if Mr. X is off?
- What does “morning” mean?

- Yes, as I said.
- Of course.
- No, at weekend the entrance remains closed.
- Clearly, it is then closed as well.
- Mr. X opens the door in this case.
- Then, a client knocks at the window to tell that the door is closed.
- …

**Requirements Engineer**

**Stakeholder**

## Your goal when writing requirements

1. **Validity:** Correct and complete requirements.
2. **Usability:** Useful requirements that make the life of your colleagues easier.

Usability contains in particular:

**Efficient** and **effective** understanding:

- Your readers shall read each requirement once.
- And share the same understanding afterwards.

# Natural Language Specification

- Used since 1950s
- **Pros:** expressive, intuitive, universal
- **Cons:** vague, ambiguous, interpretation depends on reader's background

**Style Guidelines**

- one or two short sentences of natural language
- one message per sentence
- use active voice (e.g., “the system …”)
- consistent use of language (i.e., avoid synonyms)
- use *shall* for mandatory and *should* for desirable requirements
- use text highlighting
- avoid jargon, abbreviations, acronyms
- provide a rationale

# Structured Specifications

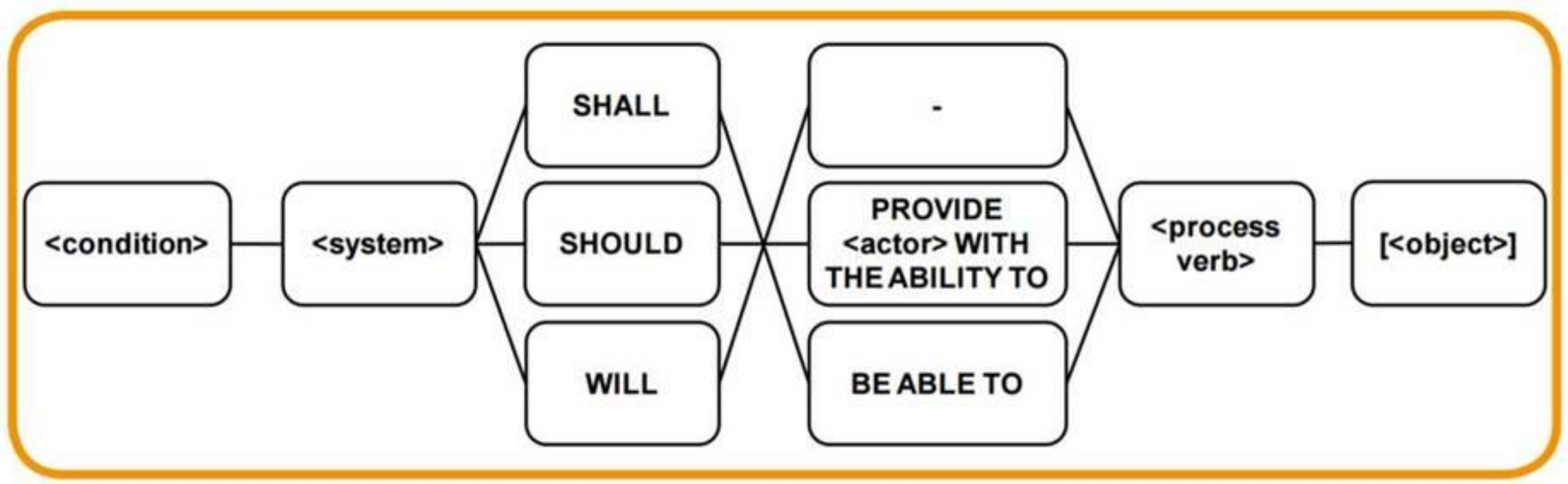
- Use of templates rather than free-form text
  - Templates may enforce structure or formulation
- **Pros:** same as before
- **Cons:** similar as before, but less variability

**Example (Structure Template)**

- **Function:** *Exchange of IDs*
- **Description:** *Two devices send their IDs to each other.*
- **Precondition:** *Devices are within a range of 2m for at least 15 minutes.*
- **Postcondition:** *IDs are stored in the local database.*
- **Rationale:** *Contact tracing requires to temporarily identify the device of contacts.*

# NL Sentence Patterns

Unified and standardized structures for writing requirements



**R1:** If the switch is pressed “up”, the system shall close the window.
**R2:** If the switch is pressed “down”, the system shall open the window.

Die Sophisten: Schablonen für alle Fälle.
https://www.sophist.de/fileadmin/SOPHIST/Puplikationen/Broschueren/SOPHIST_Broschuere_MASTeR.pdf

# NL Sentence Patterns

Easy Approach to Requirements Syntax (EARS)

| **Pattern Name** | **Pattern** |
|---|---|
| Ubiquitous | The <system name> shall <system response> |
| Event-driven | WHEN <trigger> the <system name> shall <system response> |
| State-driven | WHILE <precondition> the <system name> shall <system response> |
| Option | WHERE <feature is included> the <system name> shall <system response> |
| Unwanted behavior | IF <trigger> THEN the <system name> shall <system response> |
| Complex | <Combinations of pattern> e.g., WHILE <precondition> WHEN <trigger> the <system name> shall <system response> |

Mavin, Wilkinson, Gregory, Uusitalo:
“Listens Learned (8 Lessons Learned Applying EARS)” International Requirements Engineering Conference, 2016

## Use Cases

# Use Case

## Use Case

A use case is a list of actions or event steps typically defining the interactions between an actor and a system to achieve a goal. The actor can be a human or another external system.

## Use Case Content

**Title** - an *active-verb goal* phrase that names the goal of the primary actor

**Actors** – anyone or anything that performs a behavior (who is using the system)

**Preconditions** – what must be true or happen before the use case runs.

**Postconditions** – what must be true or happen after the use case runs.

**Triggers** – this is the event that causes the use case to be initiated.

**Main success scenario** [Basic Flow] – use case in which nothing goes wrong.

**Alternative paths** [Alternative Flow] – these paths are variations of the main scenario (errors, exceptions, alternatives).

Universität zu Köln

# Use Case – Example (Book-a-Mensa app)

| | | |
|---|---|---|
| Title | Book seat | |
| Actor(s) | Customer | |
| Preconditions | Customer has an account for the app | |
| Trigger | Customer opens booking page | |
| | **Step** | **Description** |
| Main success scenario | 1 | **Customer:** Enters login data |
| | 2 | **System:** Verifies authority (→ *UC: verify authority)* |
| | 3 | **System:** displays an overview of available seats |
| | 4 | **Customer:** Selects a seat |
| | 5 | **System:** Confirms the seat reservation via email |
| Alternative paths | 2b1 | [User has entered incorrect password]<br>**System:** Shows error message |
| | 2b2 | **Student**: Confirms error message<br>→ Return to step 1 |
| | 3a | [No seats available]<br>**System:** Shows information message<br>→ Use Case ends |

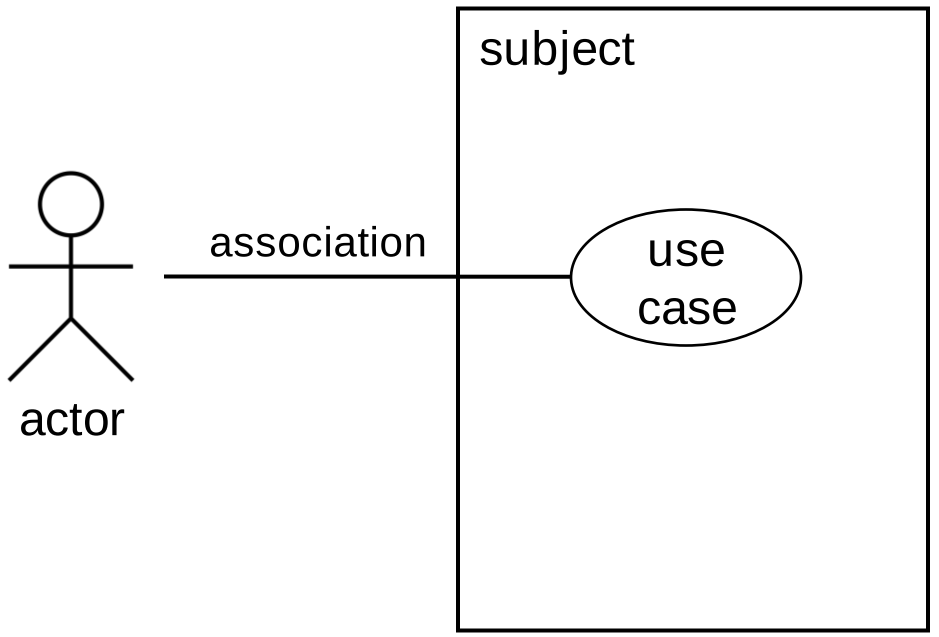
# Use Case Diagrams

**Use Case Diagram**

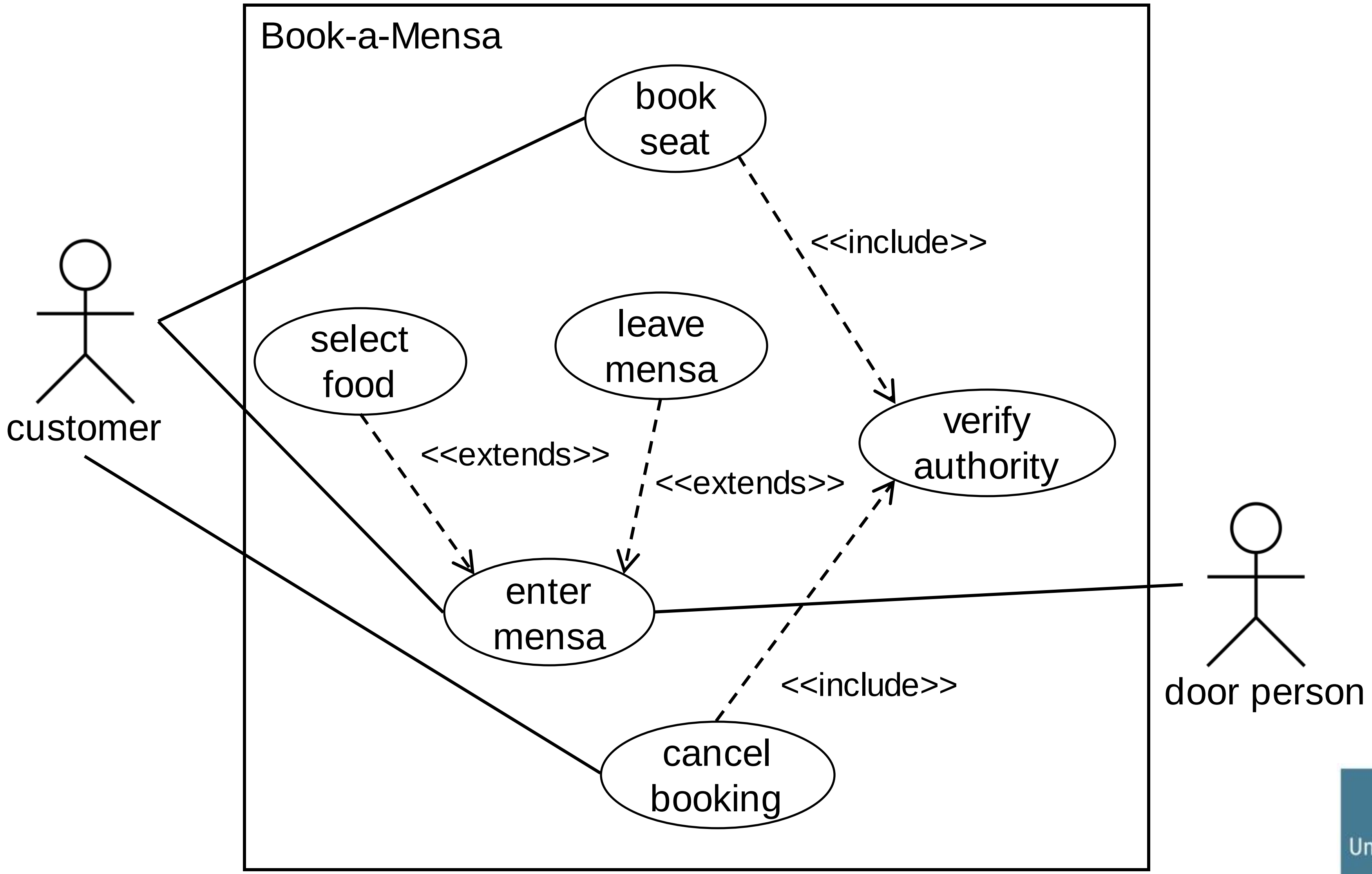
Use case diagrams are a means to capture the requirements of systems, i.e., what systems are supposed to do. The key concepts specified in this diagram are actors, use cases, and subjects:

- Each **subject** represents a system under consideration to which the use case applies. (System)
- Each users and any other system that may interact with a subject is represented as an **actor**. (Akteur)
- A **use case** is a specification of behavior in terms of verb and noun. (Anwendungsfall)

[adapted from UML v2.5.1]



# Use Case Diagrams: Example



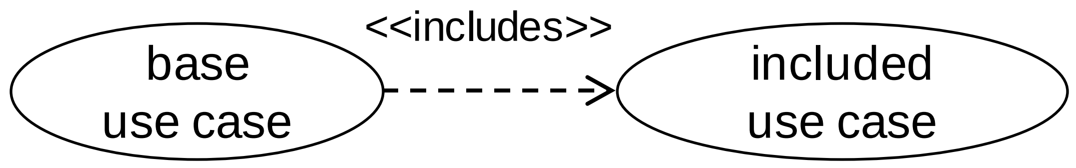
# Include and Extends Relationships

## Include Relationship

**Motivation:** make common parts of multiple use cases explicit

**Relationship:** *a base use case* may define an include relationship to an *included use case*

**Meaning:** included use case is always executed when the base use case is

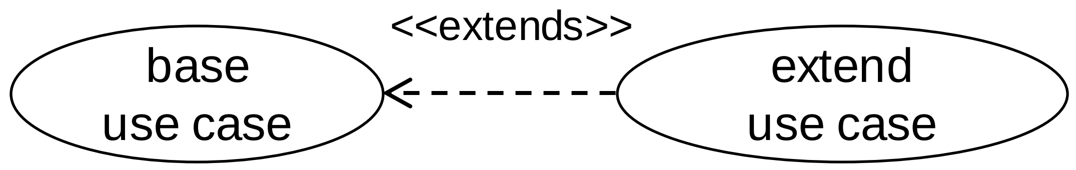


## Extends Relationship

**Motivation:** make explicit that some use cases only happen under certain circumstances

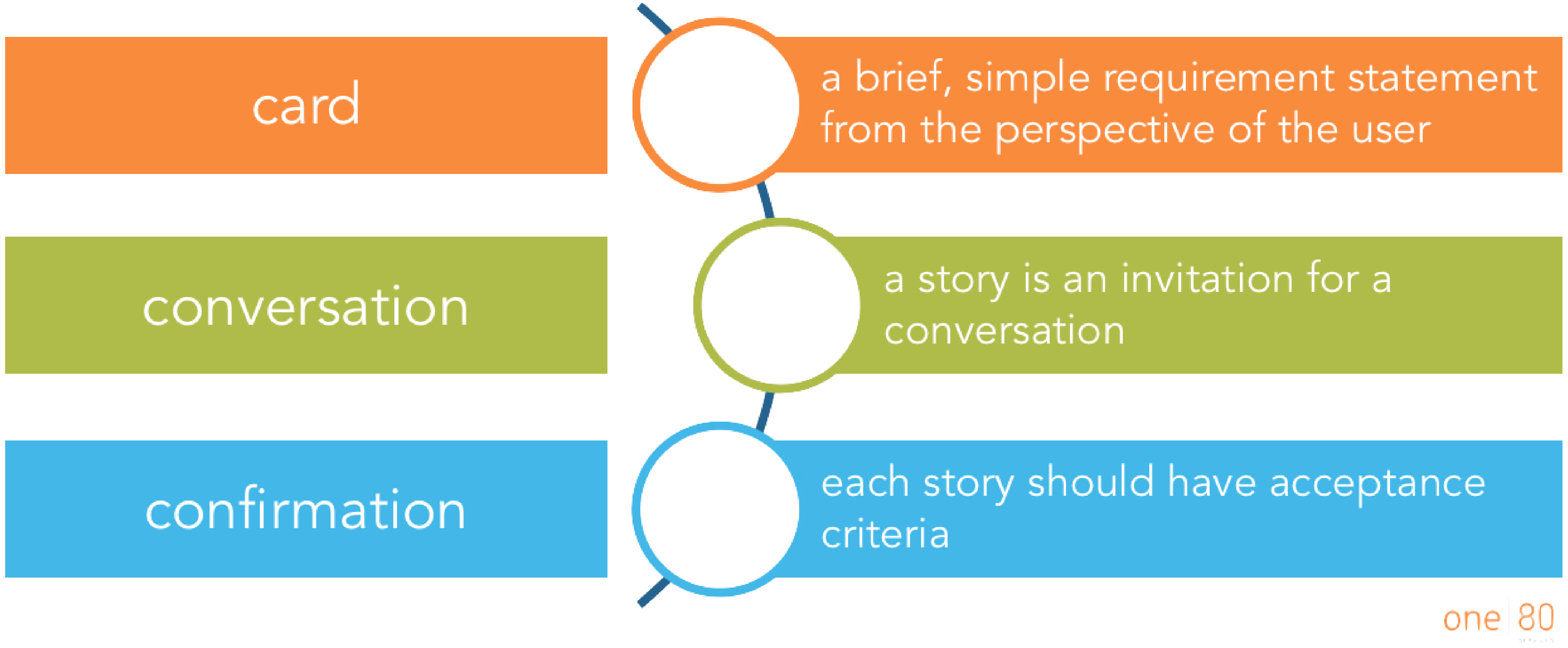
**Relationship:** an *extend use case* may define an extend relationship to a *base use case*

**Meaning:** when the base use case is executed, the extend use case may or may not be executed



# User Stories

# User Stories



| | |
|---|---|
| card | a brief, simple requirement statement from the perspective of the user |
| conversation | a story is an invitation for a conversation |
| confirmation | each story should have acceptance criteria |

Source: http://one80services.com/user-stories/writing-good-user-stories-hint-its-not-about-writing/

## Writing User Stories

- User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

- Three components that ensure that:
  1. we have someone who wants that functionality ("As a user …"),
  2. we know what the user wants ("… I want …") and
  3. we understand the why, i.e., the reason or motivation ("… so that ….")

As a <type of user>, I want <some goal> so that <some reason>.

## The Card: Front-side

Value: high

Effort: 20 SP

As a skier, I want to pass the chairlift gate so that I get access without presenting, scanning or inserting a ticket at the gate.

Author: Dan Downhill

Date: 2013-09-20

ID: S-18

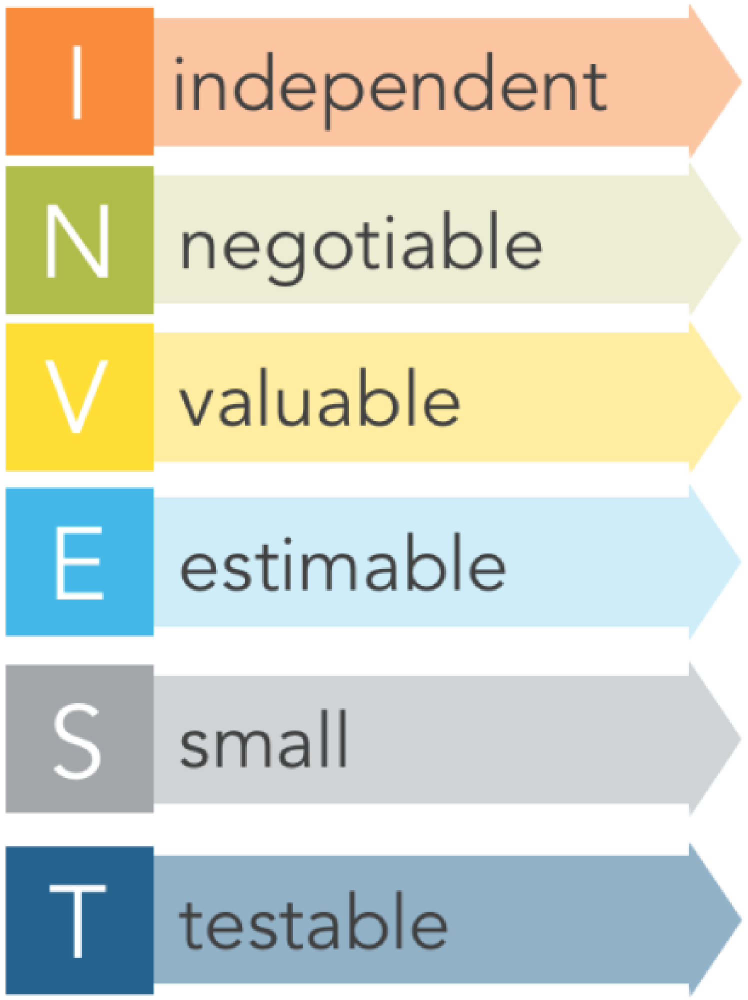
## The Card: Back-side

Acceptance criteria:

- Recognizes cards worn anywhere in a pocket on the left side of the body in the range of 50 cm to 150 cm above ground
- If card is valid: unlocks turnstile and flashes a green light for five seconds or until the turnstile is moved
- If card is invalid: doesn’t unlock gate and flashes a red light for five seconds
- Time from card entering the sensor range until unlock and flash red or green is less than 1.5 s (avg) & 3 s (max)
- The same card is not accepted twice within an interval of 200 s

Universität zu Köln

# When is a User Story good?



| | | |
|---|---|---|
| I | independent | Should be independent of other user stories (in the same sprint); it should not depend on another story being implemented first. |
| N | negotiable | Should be negotiable. Describes the essentials, not the details. |
| V | valuable | Should be valuable for the customer or user. |
| E | estimable | Effort of implementation should be estimable for the team. |
| S | small | Should be implementable in one sprint. |
| T | testable | Implementation should be verifiable. |

Source: http://one80services.com/user-stories/writing-good-user-stories-hint-its-not-about-writing/

# Requirements Validation

## Requirements Validation

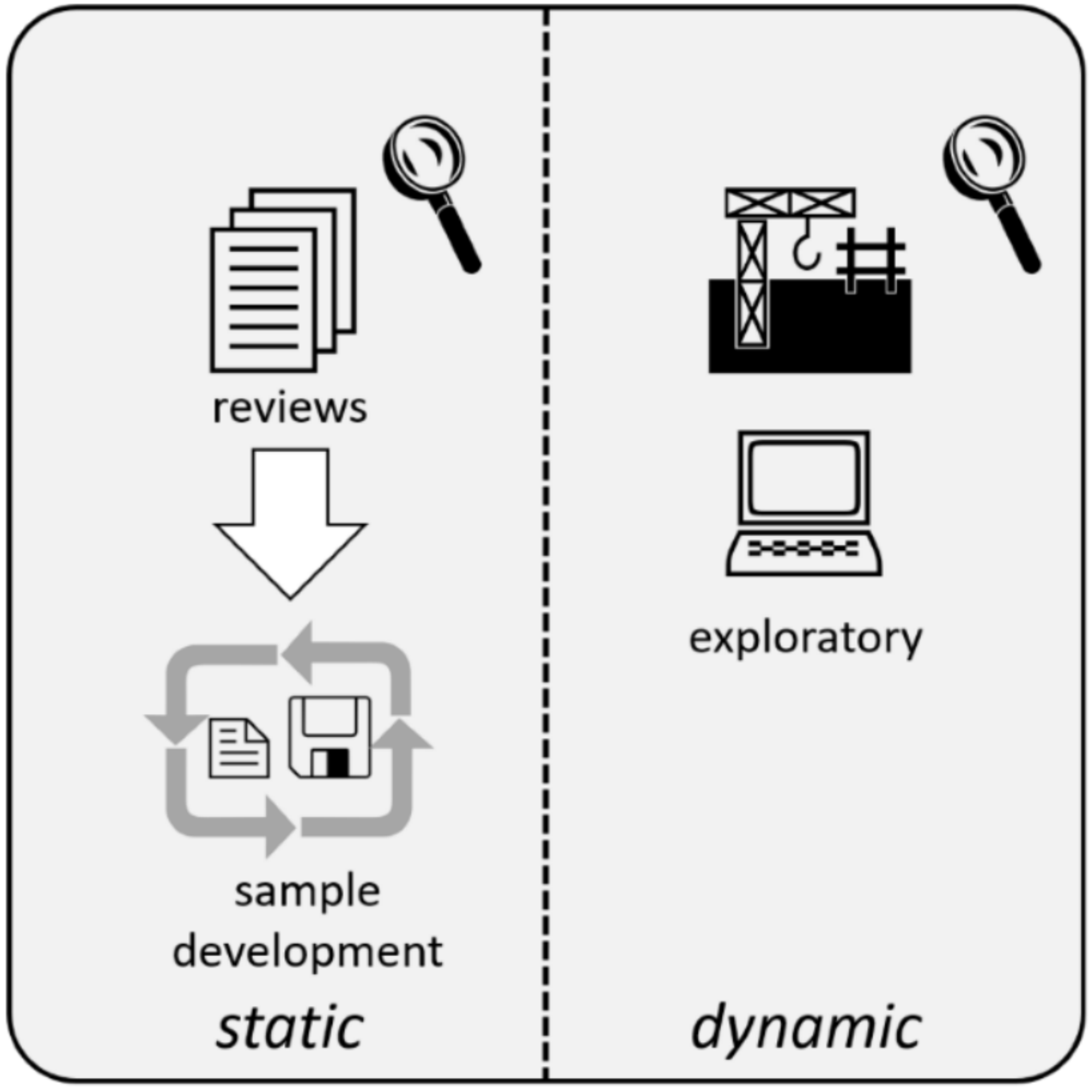
- **Validity checks:** Do requirements (still) reflect real needs?
- **Consistency checks:** Are there contradictory/redundant requirements?
- **Completeness checks:** Are all functions and constraints documented?
- **Realism checks:** Is it feasible within budget and schedule?
- **Verifiability checks:** Can we test the requirements?

**Motivation**

The later problems with requirements are detected, the more costly it will be.

# Requirements Validation Techniques

- Review Techniques
  - Visual study of early and intermediate work products
- Sample development
  - Try to produce intermediate work products (e.g., designs, code, test cases, manuals) based on a set of requirements.
- Exploratory Techniques
  - Hands-on experience with an intermediate version of (part of) the system under development



# RE Conclusions

# Requirements Engineering: Conclusions

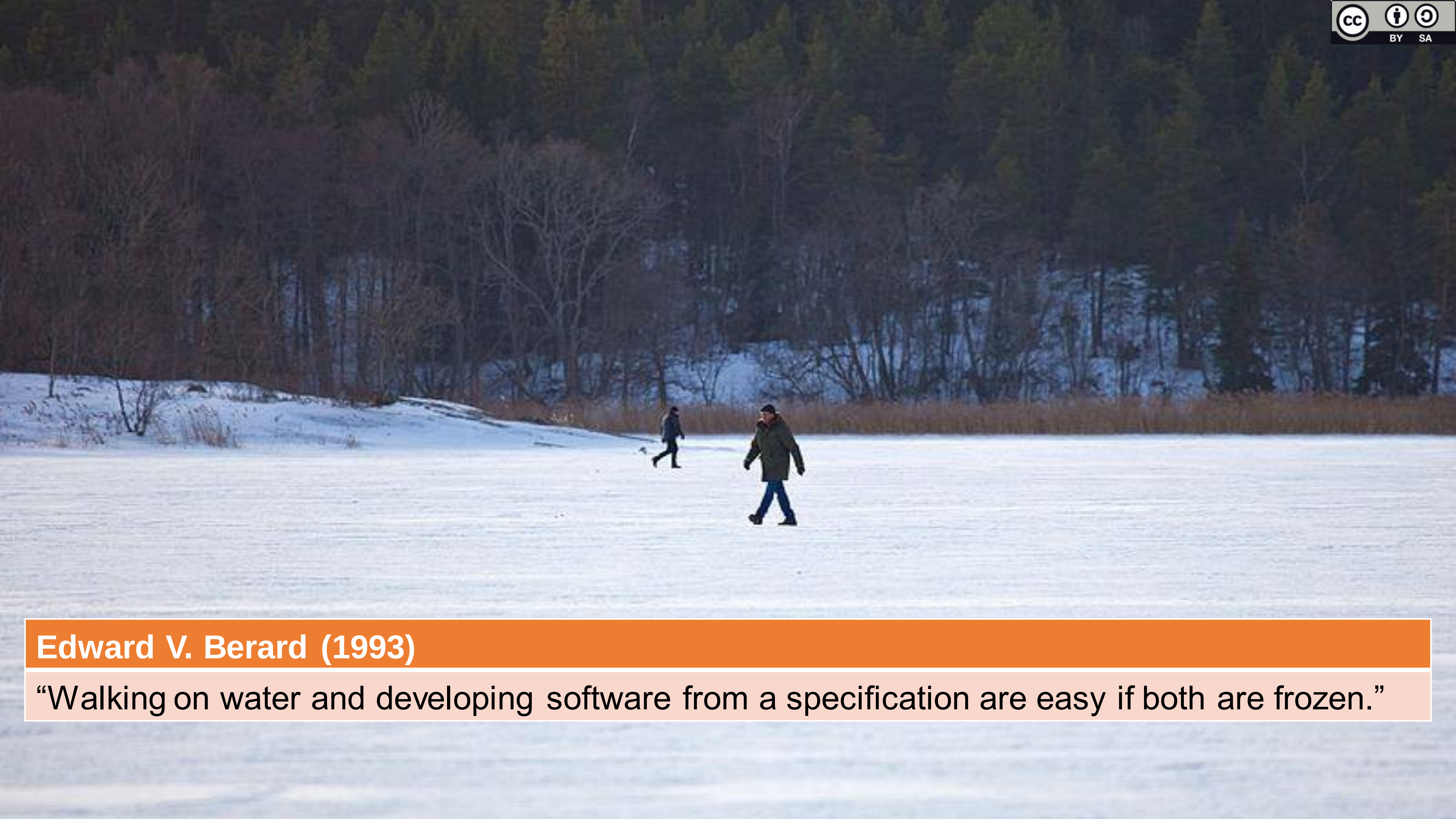
- **Stakeholders are key**
- **Validate** your requirements **early and frequently**
- **Work value-oriented**:
  - Cost and benefit of requirements need to be in balance
  - Concentrate on the essential – don’t just collect tons of detailed requirements
- **Work risk-driven**: the more risk, the more extensive and precise requirements specifications are necessary
- **Intertwining of requirements and design is natural** – you’ll need to live with it

## Requirements Engineering: Conclusions

- **Situate your system in its context**
  - Value is only created when using systems in their real-world context – so you need to know this context
  - Elicit and document domain assumptions and constraints
- **No discovery**: Requirements must be elicited with serious endeavor, they can't be just discovered
- **Strive for innovation**: just automating what we have today is not enough
- **You are not the stakeholders' voice recorder** – elicit and *design* requirements that make stakeholders excited

# Requirements Engineering: Conclusions

- **Control requirements evolution** – otherwise requirements evolution will control you
- **No universal language or method**: You'll need to use a variety of practices and languages
- **Specifying is not programming**: Skip all technical details which are not part of the problem
- **Finally: make it fun.** Nobody likes boring tasks. Make RE a *fascinating expedition* into the *unknown*, to places where the *desirable and the doable meet* and eventually merge into *exciting new opportunities*.



## Edward V. Berard (1993)

“Walking on water and developing software from a specification are easy if both are frozen.”