Course "Softwaretechnik"

Requirements Elicitation (Anforderungserhebung)

Lutz Prechelt, Steve Easterbrook

Freie Universität Berlin, Institut für Informatik

- Requirements and Requirements Engineering
  - Types and kinds of requirements
- Conventional vs. agile
- Specifications and validation
- Requirements Elicitation
  - Tasks, difficulties
- Elicitation techniques
  - Methods
  - Representatios

partially based on two sets of slides by Steve Easterbrook, University of Toronto

Where are we?: Taxonomie "Die Welt der Softwaretechnik"

Welt der Problemstellungen:

- **Produkt (Komplexitätsprob.)**
  - **Anforderungen (Problemraum)**
  - Entwurf (Lösungsraum)
- Prozess (psycho-soziale P.)
  - Kognitive Beschränkungen
  - Mängel der Urteilskraft
  - Kommunikation, Koordination
  - Gruppendynamik
  - Verborgene Ziele
  - Fehler

Welt der Lösungsansätze:

- Technische Ansätze ("hart")
  - Abstraktion
  - Wiederverwendung
  - Automatisierung
- **Methodische Ansätze ("weich")**
  - **Anforderungsermittlung**
  - Entwurf
  - Qualitätssicherung
  - Projektmanagement

Where are we?: Anforderungsermittlung

- Einsicht: Man darf sich nicht auf intuitiven Eindruck darüber verlassen, was gebaut werden sollte
  - sondern sollte die Anforderungen systematisch ermitteln
- Prinzipien:
  - **Erhebung** der Anforderungen bei allen Gruppen von Beteiligten
  - **Beschreibung** in einer Form, die die Beteiligten verstehen
  - **Validierung** anhand der verschriftlichten Form
  - **Spezifikation**: Übertragung in zur Weiterverarbeitung günstige Form
  - **Trennung von Belangen**: Anford. möglichst wenig koppeln
  - **Analyse auf Vollständigkeit**: Lücken aufdecken und schließen
  - **Analyse auf Konsistenz**: Widersprüche aufdecken und lösen
  - **Mediation**: Widersprüche, die auf Interessengegensätzen beruhen, einer Lösung zuführen (Kompromiss oder Win-Win)
  - **Verwaltung**: Übermäßige Anforderungsänderungen eindämmen, Anforderungsdokument immer aktuell halten

Definitions: Requirement

- What is a Requirement?
  - Something that someone needs in order to solve a problem or achieve an objective:
    - *"A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.*
      The set of all requirements forms the basis for subsequent development of the system or system component". [IEEE Std]
  - Note 1: Often, the "formally imposed document" does not exist, but there is still *somebody* wishing to be satisfied.
    - Informal requirements (actually more common)
  - Note 2: Often, what is written down in the "formally imposed document" will not really *satisfy* the system user
    - Invalid/incorrect requirements
  - Note 3: Requirements are definitions, **not facts**.
  - Note 4: "System" can be a computer system (**system req's**) or a socio-technical system (**user requirements**)

Definitions: Types of Requirements

- Functional requirements:
  - What the system does: the interactions between the system and its environment; independent from implementation
- Nonfunctional requirements:
  - Observable aspects of the system that are not directly related to functional behavior
  - e.g. performance or reliability aspects, etc.
- Safety/security requirements ("shall not" properties)
  - A kind of nonfunctional requirement:
    Behavior the system must never exhibit
  - e.g. "must be impossible to apply reverse thrust in mid-flight"
- Constraints ("Pseudo requirements"):
  - Imposed by the client or environment in which the system operates
  - Often concern the technology to be used (language, operating system, middleware etc.)

nonfunctional requirements

Types/kinds of requirements, a second view

- Ingenious requirements
    - incredibly valuable
  - "A cell can contain a value or a formula referring to other cells"
  - <a href=http://www.example.com/path/doc.html>
  - "SMS, sent to the public"
- Fundamental requirements
    - Not as easy as they seem
  - "The ticket machine can sell any type of ticket"
- Normal requirements
    - Can be useful, or less so
  - "The first step is selecting the route(s)"
- Usability requirements
    - tough to make concrete
  - "85% of passengers must finish their first purchase within 50 seconds"
- Detail requirements
    - super important for some kinds of software
  - eGK: SGB V, § 291 a Abs. 4 Satz 1 about groups having eRezept data access
- Usability detail requirements
    - can make a lot of difference
  - web form: "After submission, the Submit button will be deactivated"

Definitions: Requirements Engineering (RE)

- Requirements Elicitation is part of Requirements Engineering

- Requirements Engineering (RE):

"[...] **Requirements Engineering** is the branch of systems engineering concerned with real-world goals for, services provided by, and constraints on software systems. Requirements Engineering is also concerned with the relationship of these factors to precise specifications of system behaviour and to their evolution over time and across system families..." [Zave94]

"[...] RE is concerned with identifying the purpose of a software system, and the contexts in which it will be used." [RE'01 CfP]

Requirements Engineering process: 4 steps

1. Understand the problem
   - Requirements Elicitation
   - understand the context and the goals in the user's terms

2. Describe the problem (often in writing)
   - Requirements Specification
   - describe what the SW must do to reach the goals

3. Attain agreement on the problem
   - Requirements Validation
   - find gaps, mistakes, and inconsistencies in the requirements
   - includes conflict resolution, negotiation

4. Maintain the agreement
   - Requirements Management
   - negotiate and decide on changes of the specification

Source: *Adapted from Loucopoulos & Karakostas, 1995, p20 and Blum, 1992* [5]

Conventional vs. Agile

Views of Requirements Engineering: Conventional vs. Agile processes

- Requirements are defined before SW development
- Reqs are spelled out in writing precisely and in detail
- Reqs are binding obligations for the tech team

- Requirements are collected and modified all the time
- Reqs are communicated in whatever form works best (accurate, efficient)
- Reqs are opportunities for benefit generation

What's better depends *a lot*
on context!

RE step 3: Attain agreement: Conflict is natural and ubiquitous

- Even the most cooperative stakeholders ("Beteiligte") will inevitably have conflicts

- Conflict resolution is a core activity of RE

ENTSCHULDIGUNG, ABER ESSEN SIE DAS NOCH?
WARUM FRAGEN SIE? SIE WOLLEN DOCH NICHT ETWA... SIE WÜRDEN DOCH NICHT... OH GOTT.
WWW.NICHTLUSTIG.DE

[5]

RE step 3: Attain agreement: Requirements Validation

- Even without conflict, requirements validation is a critical step in the development process
  - after requirements engineering or requirements analysis
  - and again at delivery (conventional view)
- Requirements validation criteria:
  - Correctness:
    - The requirements accurately represent the client’s view.
  - Completeness:
    - All possible scenarios in which the system can be used are described, including exceptional behavior by the user or the system
  - Consistency:
    - No functional or nonfunctional requirements contradict one another
  - Feasibility/Realism:
    - Requirements can realistically be implemented and delivered
  - Traceability: (at delivery only)
    - It is possible to trace each system function to a corresponding (set of) functional requirement(s)

RE step 4 (in conventional view): Requirements Management

- Problem with requirements validation:
  Requirements change during and after elicitation

- Large projects need tool support to manage requirements:
  - Store requirements in a shared repository
  - Provide multi-user access
  - Automatically create a system specification document from the repository
  - Allow change management
  - Provide traceability throughout the project lifecycle

- e.g. IBM Rational DOORS or
  an appropriate issue tracker tool

Requirements Engineering differences in Agile processes

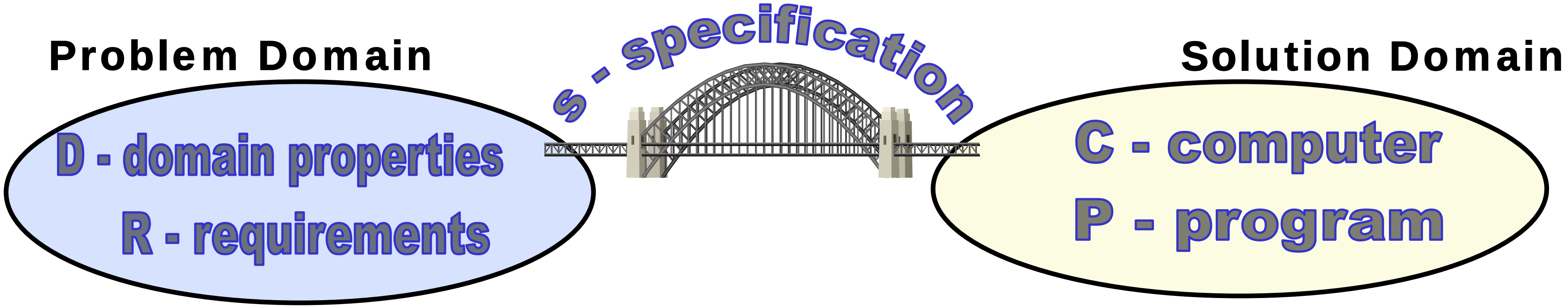
1. Understand the problem
   - Use whatever techniques appear to work
   - In particular, collect feedback from software users

2. Describe the problem
   - Writing is a lot of work; results will often be misunderstood.
   - Prefer oral communication where possible.

3. Attain agreement on the problem
   - Find oversights and inconsistencies also by trying things out
   - *Some* conflicts can be resolved by deferring and making smaller steps

4. Maintain the agreement
   - Changing things is the normal state of existence!

Requirements and specifications



Problem Domain

D - domain properties

R - requirements

S - specification

Solution Domain

C - computer

P - program

- **Domain Properties** are properties in the problem domain that are true whether or not we ever build the proposed system
- **Requirements** are properties in the problem domain that we wish to be made true by delivering the proposed system
- **A specification** is a description of the behaviors of the program in the solution domain that the program must have in order to meet the requirements
  - The system specification (system requirements), not to be confused with a statement of the requirements themselves, the requirements specification (user requirements)

Source: *Adapted from Jackson, 1995, p170−171*

Validation vs. Verification

- Verification checks the equivalence of different formal representations
- Validation checks if a system fulfills the actual expectations in the real world

- **Verification** criteria:
  - Does the Program running on a particular Computer satisfy the Specification?

- **Validation** is more comprehensive, it implicitly also checks:
  - Did we understand all the important Requirements?
  - Did we understand all the relevant Domain properties?

Validation example

- Requirement R:
  - "Reverse thrust shall only be enabled when the aircraft is moving on the runway"
- Domain Properties D:
  - Wheel pulses are on if and only if wheels are turning
  - Wheels are turning if and only if aircraft is moving on runway
- Specification S:
  - Reverse thrust is enabled if and only if wheel pulses are on
- S + D imply R
  - But what if the domain model D is wrong?

(Do you recognize the example?)

In radical-design requirements, we may easily misunderstand domain proberties.

***Source:** Adapted from Jackson, 1995, p172*

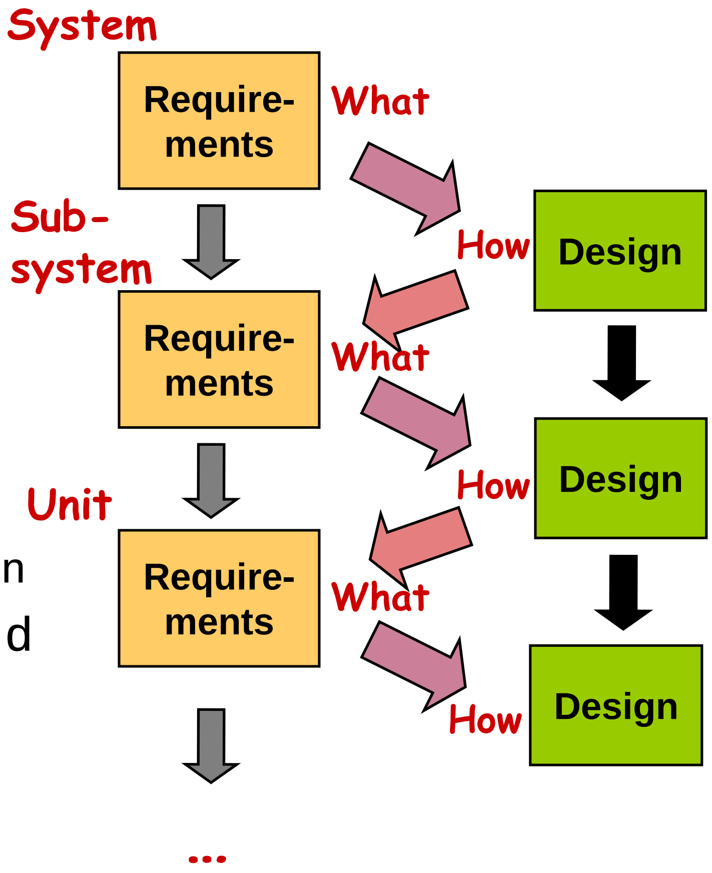
Another validation example

- Requirement R:
  - "The database shall only be accessible by authorized personnel"
- Domain Properties D:
  - Authorized personnel have passwords
  - Non-authorized personnel do not have passwords
- Specification S:
  - Access to the database shall only be granted after the user types an authorized password
- S + D imply R
  - But what if the domain assumptions are wrong?
  - A sensible SW engineer will question all domain assumptions

Source: *Adapted from Jackson, 1995, p172*

What vs. How

- "Requirements should specify **what** without specifying **how**"
  - But this is not always easy to distinguish:
    - What does a car do vs. a bike?
      - (Don't mention the motor: 'how'!)
    - The 'how' at one level of abstraction forms the 'what' for the next level
- A suitable distinction
  - 'What' refers to a system’s purpose
    - it is **external** to the system
    - it is a property of the application domain
  - 'How' refers to a system’s structure and behavior
    - it is **internal** to the system
    - it is a property of the solution domain
  - *Interfaces* are boundaries between 'What' and 'How'



Source: *Adapted from Jackson, 1995, p207*

What is a System?

Definition of a System:

- Some part of reality that can be observed to interact with its environment
  - Separated from its environment by a boundary
    - Boundary may be difficult to decide: "soft" system
  - A system receives inputs from the environment and sends outputs to the environment
  - Many systems have a control mechanism
  - Most systems have interesting emergent properties

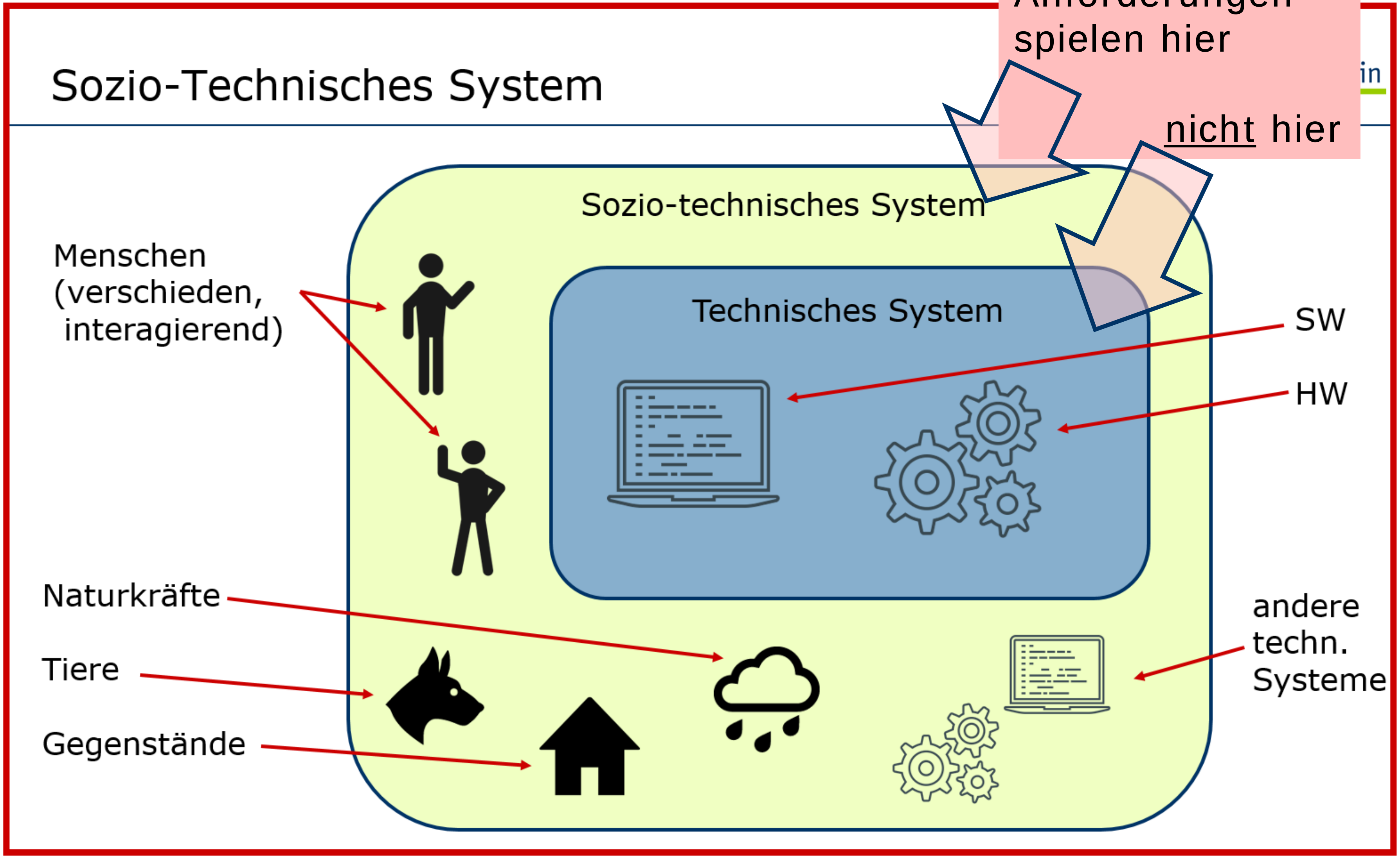
- Examples:
  - cars, cities, houseplants, rocks, spacecraft, buildings, weather,...
  - operating systems, DBMS, The Sims, Instagram, the Internet
- Non-examples (there aren’t many!):
  - numbers, truth values, letters

Source: *Adapted from Wieringa, 1996, p10*

Most systems are "soft": socio-technical

- The software we will eventually write is not **"the system"** with respect to requirements engineering
  - The software is the system only in the solution domain
  - but not in the problem domain
- Rather, other things are also part of the system in the problem domain:
  - the people using the software,
  - the ways in which they use it,
  - many other environmental factors
- This larger system we need to understand during requirements elicitation
  - Rule of thumb: If people are involved in any way, never confuse the software with the system
  - Remember "Auswirkungen d. Informatik"?: underground train with taped-down "GO" button?
    - Very simple software, but a surprising system

Siehe "Auswirkungen der Informatik"



Sozio-Technisches System

Sozio-technisches System

Technisches System

Menschen
(verschieden,
interagierend)

SW

HW

Naturkräfte

Tiere

Gegenstände

andere
techn.
Systeme

Anforderungen
spielen hier

nicht hier

[5]

Requirements Elicitation

- Starting point: Some notion that there is a "problem" that needs solving
  - something negative to get rid of
  - or an opportunity to be exploited

- The requirements engineer must:
  - become enough of an expert in the problem domain to
  - identify the problem and opportunity and
  - elicit enough knowledge to analyze requirements for
  - validity, consistency, and completeness

W6H

The journalist's technique:

What?

Where?

Who?

Why?

When?

How?

(Which?)

Identifying
the problem and opportunity

- Which problem needs to be solved?
  - identify problem Boundaries
- Where is the problem?
  - understand the Context/Problem Domain
- Whose problem is it?
  - identify Stakeholders (Betroffene, Beteiligte)
- Why does it need solving?
  - identify the stakeholders' Goals
- How might a software system help?
  - collect some Scenarios
- When and how does it need solving?
  - identify Development Constraints
- What might prevent us solving it?
  - identify Feasibility and Risk

Very useful slide

W6H

The journalist's technique:

What?
Where?
Who?
Why?
When?
How?
(Which?)

Difficulties of Elicitation (1)

- **Limited observability**
  - The problem owners might be too busy solving it in its current form
  - Presence of an observer may change the problem
- **Thin spread of domain knowledge**
  - It might be distributed across many sources
  - Is rarely available in explicit form
- **Bias**
  - People may not be free to tell you what you need to know
    - Political climate & organizational factors
  - People may not want to tell you what you need to know
    - The outcome will affect them, so they may try to influence you (hidden agendas)
- There will be **conflicts** between different sources
  - People have conflicting goals
  - or different understandings

Difficulties of Elicitation (2): Tacit knowledge

- **Tacit knowledge**
  (The "say-do" problem)
  - Experts are not aware of what they know and cannot introspect reliably
  - Can solve any instance,
    but cannot state a general rule

- **Representational Problems**
  - Experts don’t have the language to describe their knowledge
  - Spoken language lacks precision
  - Different knowledge representations are good for different things

- **Brittleness**
  - Knowledge is created, not extracted:
    incomplete, overly simplified

Difficulties of Elicitation (3): **Distortions**

Sender-related:

- Group think
  - Response to reactions of other experts
- Impression management
  - Response to imagined reactions of managers, clients, etc.
- Availability
  - Some data are easier to recall than others
- Underestimation of uncertainty
  - Tendency to underestimate by a factor of 2 or 3

Receiver-related:

- Misinterpretation
  - due to lack of knowledge
- Misrepresentation
  - e.g. question was yes/no,
    answer is yes/no,
    but reality is more complicated
- Anchoring
  - Contradictory data is ignored once an initial solution is available

Sender- and receiver-related:

- Inconsistency
  - Statements made earlier are forgotten

Difficulties of Elicitation (4)

- **Personal** and interpersonal **factors**

Method: Introspection

-
- Just sit down and think what the requirements may be
  - Very popular with software engineers
  - But then often in the form:
    Just sit down and think up <u>some</u> requirements
- Advantages
  - Simple, quick, cheap, no misunderstandings
- Disadvantages
  - Often not applicable ("I have no idea")
  - **Can be extremely misleading**
    - The mantra of usability people is: "Users are not like us!"

Method: Participant observation ("teilnehmende Beobachtung")

- Approach
  - Observer spends time with the subjects, joining in, long enough to become a member of the group

- Advantages
  - Highly contextualized and relatively reliable
  - Reveals details that other methods cannot
- Disadvantages
  - **Extremely time consuming!**
  - Resulting 'rich picture' is hard to analyze
  - Cannot say much about the results of proposed changes
- Watch for
  - going native!

Method: Interviews

- Types:
  - Semi-structured – agenda of fairly open questions
  - Open-ended – no pre-set agenda
- Advantages
  - Rich collection of information
- Disadvantages
  - **Interviewing is a difficult skill to master**
  - Large amount of qualitative data can be hard to analyze
- Watch for
  - All the difficulties listed above
  - Removal from context

[5]

***Source:** Adapted from Goguen and Linde, 1993, p154.*

Method: Questionnaires

- Advantages
  - Can cheaply collect information from many people

- Disadvantages
  - Presupposed answer categories lose context and provoke misrepresentations
  - Free-text answers are often highly ambiguous

- Sometimes useful, but often a dangerously simplistic idea!

Methods: Group Elicitation Techniques

- Types:
  - Joint/Rapid Application Development (JAD/RAD) Workshops
  - Focus Groups

- Advantages
  - More natural interaction between people than formal interview
  - Produces more ideas
- Disadvantages
  - Requires a highly trained facilitator
  - **Danger of Groupthink**
- Watch for
  - Dominance and submission
  - Superficial responses where detail is needed

Methods: Use user feedback

- Types:
  - Discussion in forums, bug trackers, user conference workshops
  - Talk to your user support crew
  - Monitoring data, A/B testing, etc.

- Advantages
  - "Real"
  - Makes problem domain and solution domain overlap
  - Needs fewer interpretations

- Disadvantages
  - Some input looks sensible, but is nonsense

- Watch for
  - Misunderstandings
  - Feature creep

Prefered approach in Agile processes

Method: Iterative development

- Developing in iterations (as opposed to all-in-one-go) can itself be considered a requirements elicitation technique

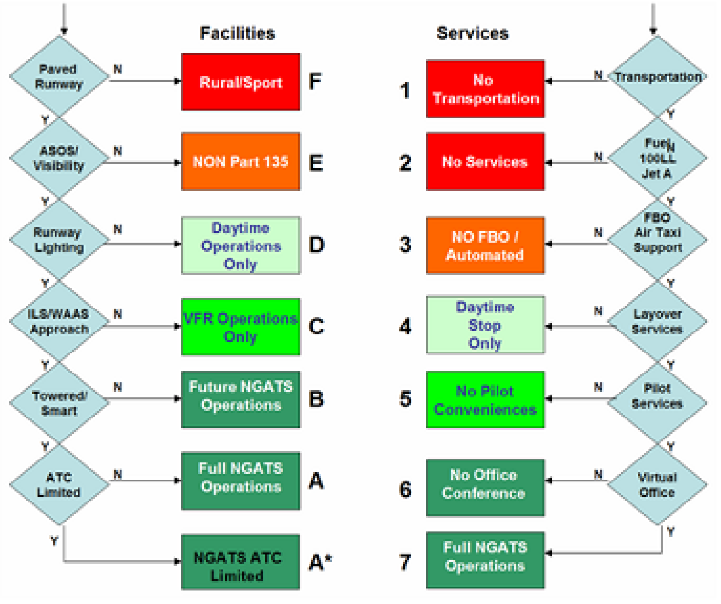
- Each iteration is an opportunity to rethink requirements
  - and one has invariably learned something from the previous iteration

Inherent approach in Agile processes

Representation-based method: Card sorting

- For a given set of domain objects, written on cards:
  - Expert sorts the cards into groups...
    - which requires an agreed-upon set of objects
  - ...then explains what the criterion was for sorting and what the groups represent
- Advantages
  - **Good for eliciting tacit knowledge**
- Disadvantages
  - Only models classification knowledge, **not performance knowledge**

Airport classification criteria proposed by CASA



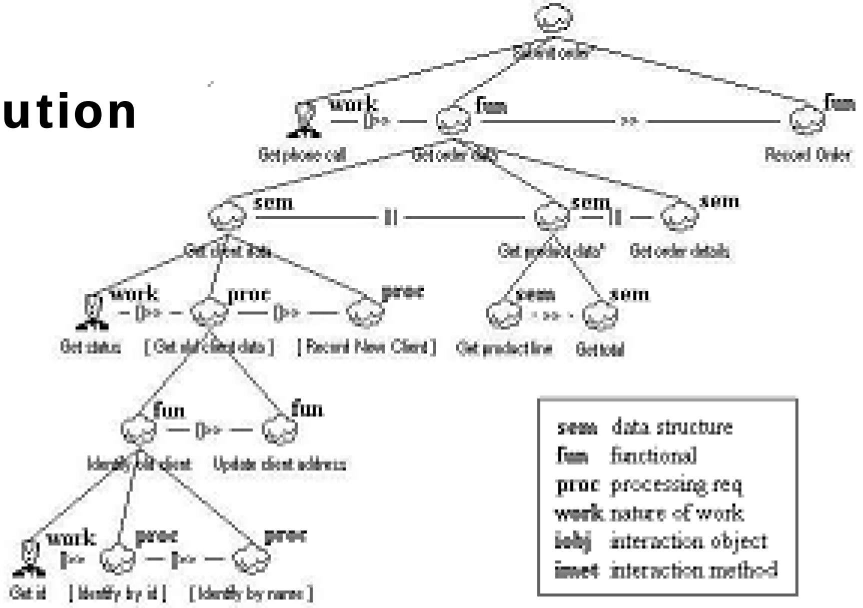
Data from CASA

Representation: Goal hierarchies

- Approach
  - Focus on why systems are constructed
  - Express the 'why' as a set of stakeholder goals
    - The top-level goal is often "save money" or "make money"
    - Use hierarchical goal refinement and impediment cross-links

- Advantages
  - Simple
  - **Sound basis for conflict resolution**

- Disadvantages
  - **Either gets very complex**
    (can lead to analysis paralysis)
    or lacks detail



Source: *Adapted from Anton, 1996.*

[5]

Representation: Scenarios

- Example sequences of interaction between actor and system
  - May be positive (required behavior)
  - or negative (an undesirable interaction)

- Advantages
  - Very natural: stakeholders tend to use them spontaneously
  - Easy to understand (low level of abstraction)

- Disadvantages
  - Lack of structure
    - but grouping them into use cases helps

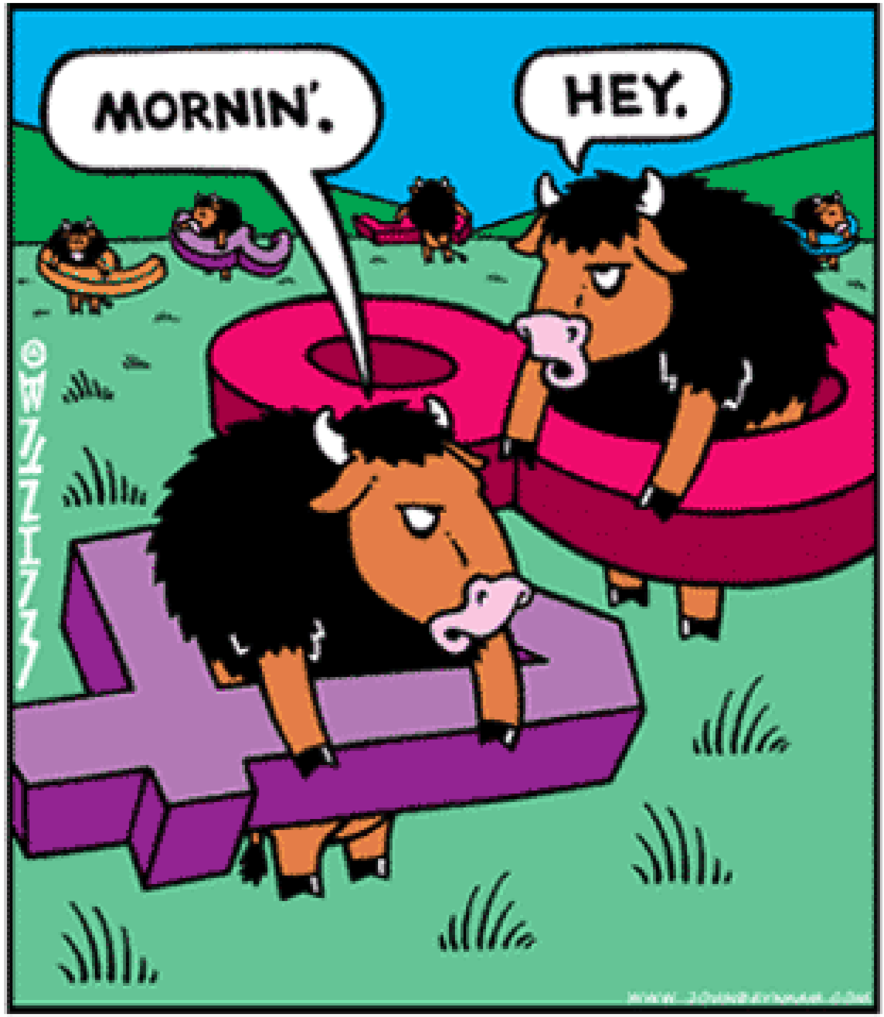
Source: *Adapted from Dardenne, 1993.*

Note: Beware of natural language!

- Natural language is easy-to-use, natural, and often appropriate for describing requirements
- But it is highly **ambiguous**!
- Example:

  "Buffalo once roamed
  the plains in large numbers"

  - Now please misunderstand this statement creatively!
    - This is going to happen to some of your natural-language requirements!



Buffalo once roamed the plains in large numbers.

Summary

- Requirements represent the goals to be reached via a software system
- A specification (written down or not) describes what the software must do in order to fulfill the requirements
  - assuming certain domain properties are met
- Requirements elicitation is the basic step of Requirements Engineering
  - others are Req. Specification, Req. Validation, and Req. Management
    - which in Agile methods get closely intertwined

- Requirements Eliciation must overcome many recurring problems
- Many different elicitation techniques should be combined

Literature

- James Robertson, Suzanne Robertson: "Mastering the Requirements Process: Getting Requirements Right", 3rd ed., Addison-Wesley 2012

- Donald Gause, Gerald Weinberg: "Exploring Requirements – Quality before Design", B&T, 1989
  - auf deutsch: "Software Requirements: Anforderungen erkennen, verstehen und erfüllen", (vergriffen)
  - http://www.geraldmweinberg.com

Thank you!