

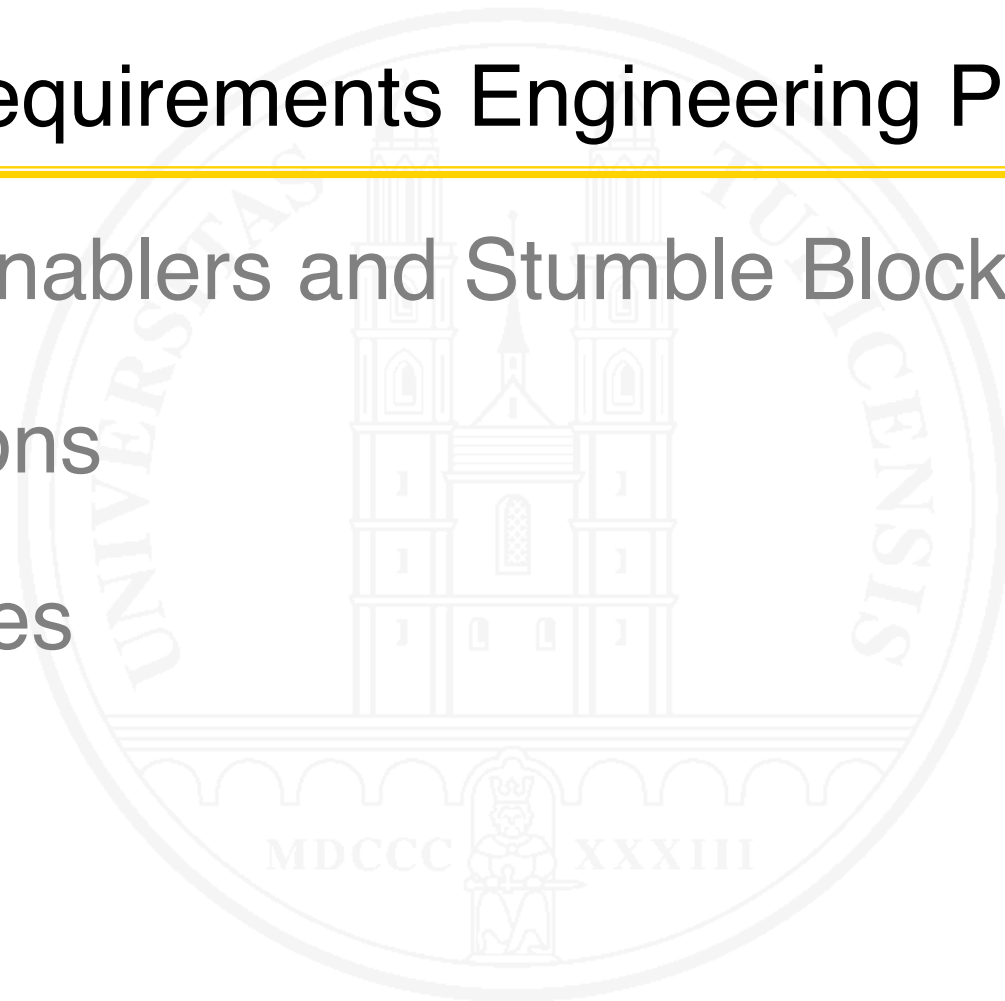
Part I: The Fundamentals

Part II: Requirements Engineering Practices

Part III: Enablers and Stumble Blocks

Conclusions

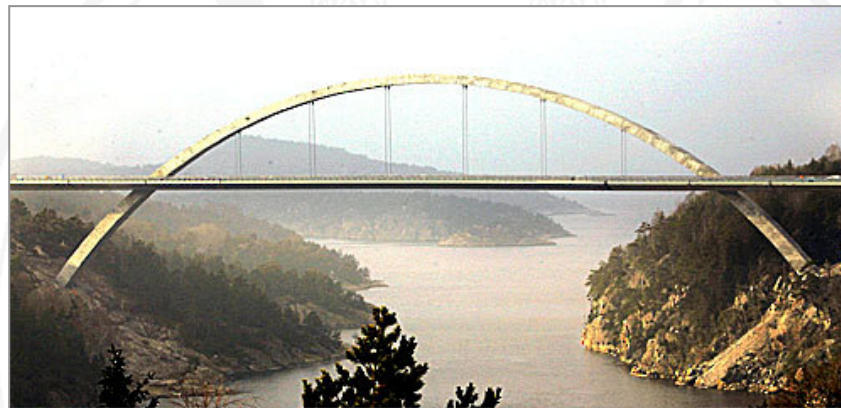
References



5 Documenting requirements

Bridging the gap:

Stakeholders



System builders

Photo © Lise Aserud / DPA

The need:

- Communicating requirements
- Having a basis for contracts and acceptance decisions

The means: A requirements specification document

5.1 Document types

Various document types, depending on RE process and specification purpose

- **Stakeholder requirements specification** (also called **customer requirements specification**)
What the **stakeholders want** (independent of any system providing it)
- **System requirements specification**
The **system to be developed** and its context
- **Software requirements specification**
If the system is a **pure software** system

Document types – 2

- **Business requirements specification**
High-level specification of business needs or goals
- **Collection of user stories and/or task descriptions**
Used in agile software development

Stakeholder requirements specification

- Written when **stakeholder needs** shall be documented before any system development considerations are made
- Typically written by **domain experts** on the **customer** side (maybe with help of RE consultants)
- If a stakeholder requirements specification is written, it **precedes** and **informs** system or software requirements specifications

System/software requirements specification

- The **classic** form of a requirements specification
- **No methodological difference** between **system** requirements specification and **software** requirements specification
- Typically written by **requirements engineers** on the **supplier** side

5.2 Aspects to be documented

Independently of any language and method, **four aspects** need to be documented:

- **Functionality**
 - **Data:** Usage and structure
 - **Functions:** Results, preconditions, processing
 - **Behavior:** Dynamic system behavior as observable by users
 - Both **normal** and **abnormal cases** must be specified

Aspects to be documented – 2

- **Performance**
 - Data **volume**
 - Reaction **time**
 - Processing **speed**
 - Specify measurable values if possible
 - Specify more than just average values

- **Specific qualities**
 - “-ilities” such as
 - Usability
 - Reliability
 - etc.

Aspects to be documented – 3

○ Constraints

Restrictions that must be obeyed / satisfied

- **Technical**: given interfaces or protocols, etc.
- **Legal**: laws, standards, regulations
- **Cultural**
- **Environmental**
- **Physical**
- **Solutions / restrictions** demanded by important stakeholders

5.3 How to document

Sample documentation standards

IEEE Std 830-1998 (outdated, but still in use)

- Three parts
- System requirements only
- Representation of specific requirements tailorable

VOLERE

- 27 chapters
- System and project requirements

Enterprise-specific standards

- Imposed by customer or given by supplier

IEEE Std 830-1998

1. Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, acronyms, and abbreviations
 - 1.4 References
 - 1.5 Overview
2. Overall description
 - 2.1 Product perspective
 - 2.2 Product functions
 - 2.3 User characteristics
 - 2.4 Constraints
 - 2.5 Assumptions and dependencies

3. Specific requirements
- Appendixes
- Index

- Variants:
Organize by
- Mode
 - User class
 - Object
 - Feature
 - Stimulus
 - Function

Project Drivers

1. The Purpose of the Project
2. Client, Customer & other Stakeholders
3. Users of the Product

Project Constraints

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

Functional Requirements

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

Non-Functional Requirements

10. Look and Feel Requirements
11. Usability and Humanity Requirements
12. Performance Requirements
13. Operational Requirements

14. Maintainability and Support Requirements

15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

Project Issues

18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
22. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

How to document – language options

Informally

- Natural language (narrative text)

Semi-formally

- Structural models
 - Interaction models
- } Typically as diagrams which are enriched with natural language texts

Formally

- Formal models, typically based on mathematical logic and set theory

General rules for requirements documentation

- Specify every requirement as a **small, identifiable unit**
- Add **metadata** such as source, author, date, status
- Build the requirements document according to some **structure template**
- Adapt the degree of detail to the **risk** associated with a requirement
- Specify **normal** and **exceptional** cases
- Don't forget **quality requirements** and **constraints**



© UFS, Inc.

Precision – Detail – Depth

Three dimensions:

How precise?

How deep, i.e., how many layers?

Dimensions influence each other:

- More precision → more detail
- More detail → more depth

How much detail?

Precision: reduce ambiguity

Restrict your language

Use a glossary

Define acceptance test cases

Quantify where appropriate

Formalize



Snoopy quantifies ... unfortunately, I have it only in German

Detail

What's better?

“The participant entry form has fields for name, first name, sex, ...”

“The participant entry form has the following fields (in this order): Name (40 characters, required), First Name (40 characters, required), Sex (two radio buttons labeled male and female, selections exclude each other, no default, required),...”

It depends.

- Degree of **implicit shared understanding** of problem
- Degree of **freedom** left to designers and programmers
- **Cost vs. value** of detailed specification
- The **risk** you are willing to take

Depth

The more precise, the more information is needed

→ Preserve readability with a hierarchical structure

“
...

4.3 Administration of participants

4.3.1 Entering a new participant

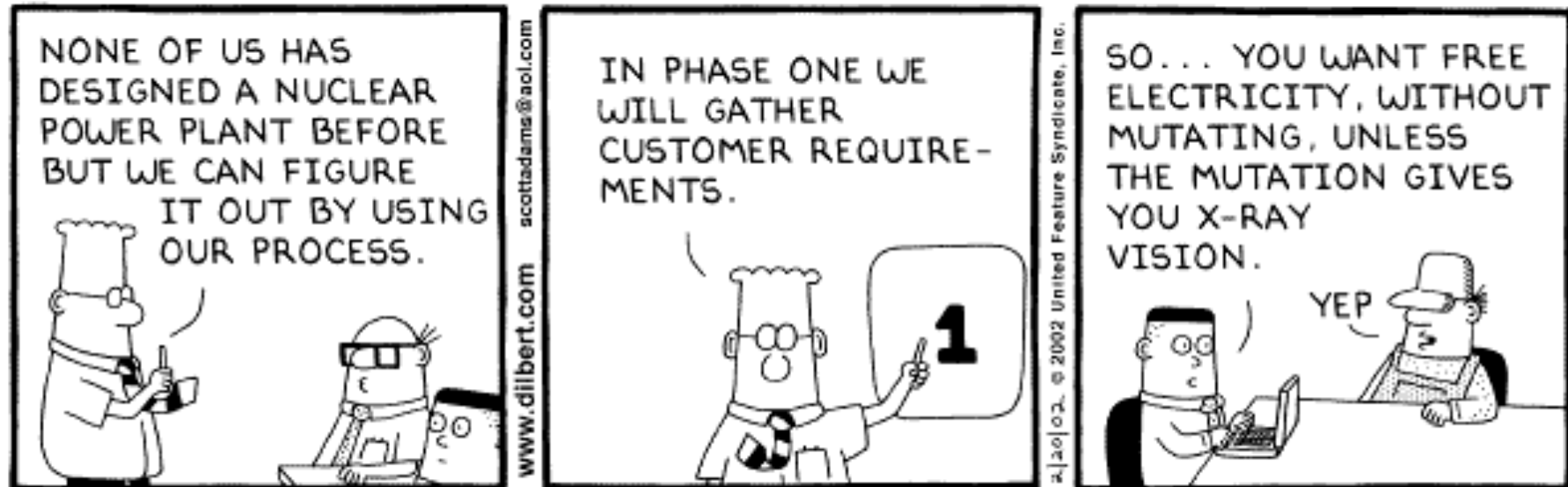
4.3.1.1 New participant entry form

4.3.1.2 New participant confirmation

4.3.2 Updating a participant record

”
....

6 Requirements Engineering processes



The principal tasks

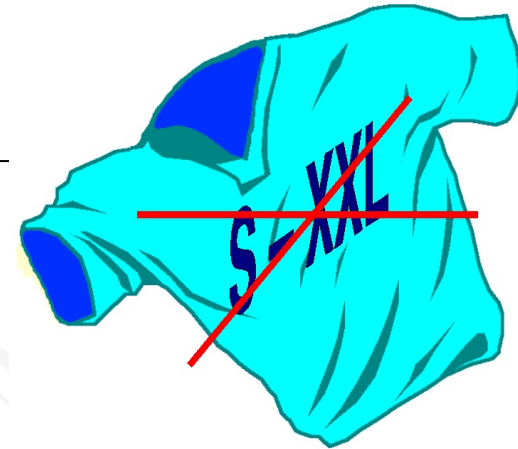
Requirements **Specification**

- Elicitation
- Analysis
- Documentation
- Validation

Requirements **Management**

- Identification and metadata
- Requirements prioritization
- Change and release management
- Traceability

No 'one size fits all' process



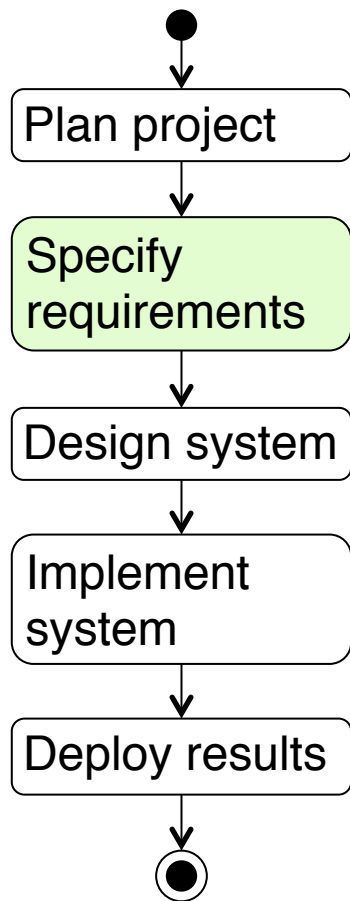
Some determining factors

- The embedding process: linear or incremental?
- Contract (prescriptive) or collaboration (explorative)?
- Can you talk with your stakeholders?
- Customer order or development for a market?
- Using COTS?

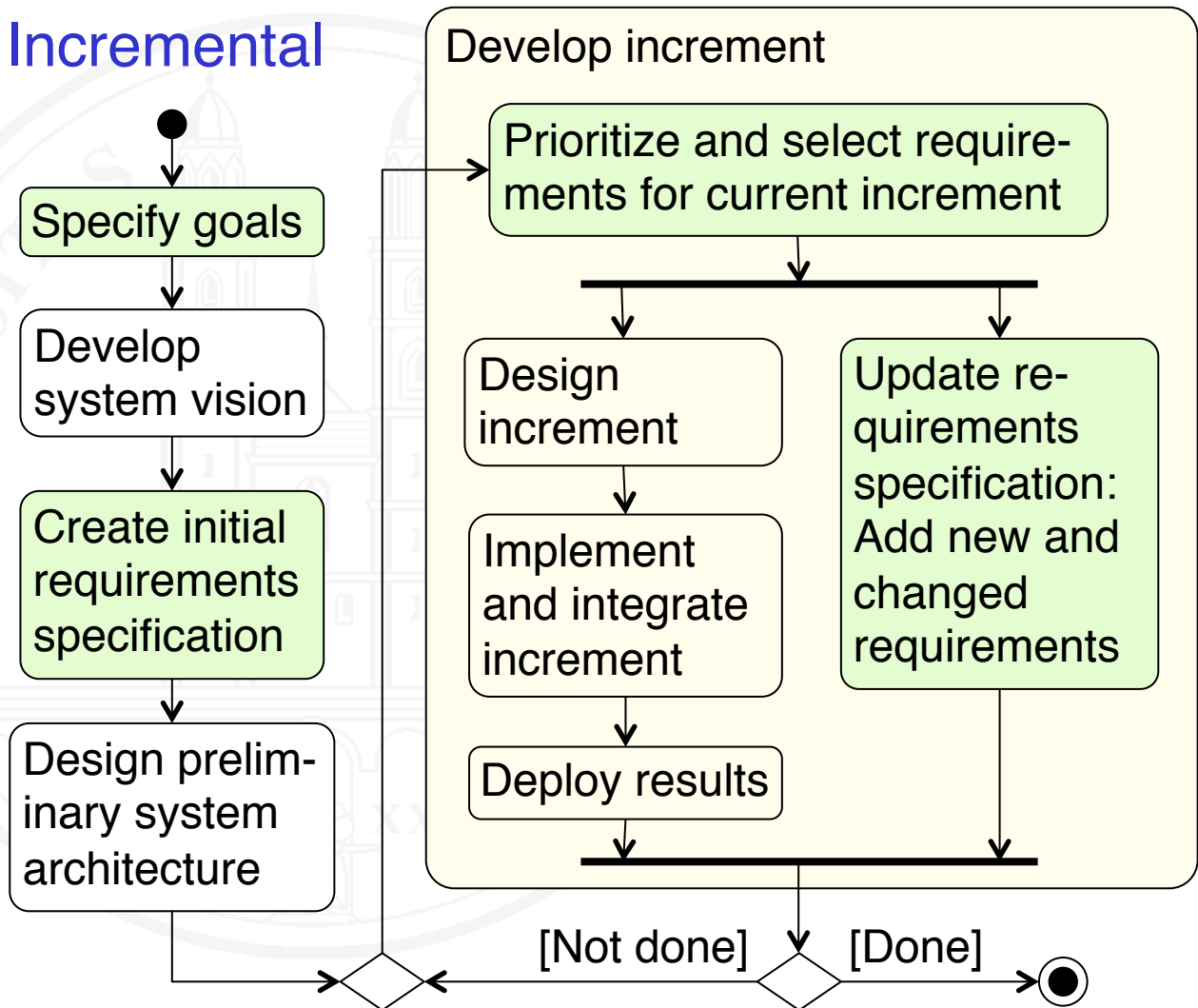
⇒ Tailor the process from some principal configuration options and a rich set of RE practices

Linear vs. incremental processes

Linear



Incremental



Linear vs. incremental processes – 2

Decision criteria

○ Linear

- Clear, stable, a priori known requirements
- Low risk (of developing the wrong system)
- Relatively short duration of project
- Complex requirements change process is acceptable

○ Incremental

- Evolving requirements
- High risk (of developing the wrong system)
- Long duration of project
- Ability to change requirements easily is important

Prescriptive – Explorative – COTS-driven

Prescriptive process

- Requirements specification is a **contract**: All requirements are binding and must be implemented
- **Functionality** determines cost and deadlines
- Needed when **tendering** design and implementation
- Development of specified system may be **outsourced**
- Frequently combined with **linear** processes

Explorative process

- Only goals known, concrete requirements have to be **explored**
- **Stakeholders** strongly **involved**, **continuous** feedback
- **Prioritizing** and **negotiating** requirements to be implemented
- **Deadlines** and **cost** constrain functionality
- Typically works only with **incremental** processes

Prescriptive – Explorative – COTS-driven – 2

COTS-driven process

- System will be implemented with **COTS** software
- Requirements must reflect **functionality** of **chosen COTS solution**
- Requirements need to be **prioritized** according to importance
- Frequently, only requirements **not covered** by the COTS solution are specified

COTS (Commercial Off The Shelf) – A system or component that is not developed, but bought as a standard product from an external supplier

Customer-specific vs. Market-oriented

Customer-specific process

- System is **ordered** by a **customer** and **developed** by a supplier **for this customer**
- **Individual persons** can be **identified** for all stakeholder roles
- Stakeholders on **customer** side are **main source** for requirements

Market-oriented process

- System is developed as a **product** for the market
- Prospective users typically **not individually identifiable**
- Requirements are specified by supplier
- **Marketing** and system **architects** are primary stakeholders
- Supplier has to **guess/estimate/ elicit** the **needs** of the envisaged customers

Typical requirements process configurations

- **Participatory:** incremental & exploratory & customer-specific
 - **Main application case:** Supplier and customer closely collaborate; customer stakeholders strongly involved both in specification and development processes
- **Contractual:** typically linear (sometimes explorative) & prescriptive & customer-specific
 - **Main application case:** Specification constitutes contractual basis for development of a system by people not involved in the specification and with little stakeholder interaction after the requirements phase

Typical requirements process configuration

- **Product-oriented:** Incremental & mostly explorative & market-oriented
 - **Main application case:** An organization specifies and develops software in order to sell/distribute it as a product
- **COTS-aware:** [Incremental | linear] & COTS-driven & customer-specific
 - **Main application case:** The requirements specification is part of a project where the solution is mainly implemented by buying and configuring COTS

Agile requirements process

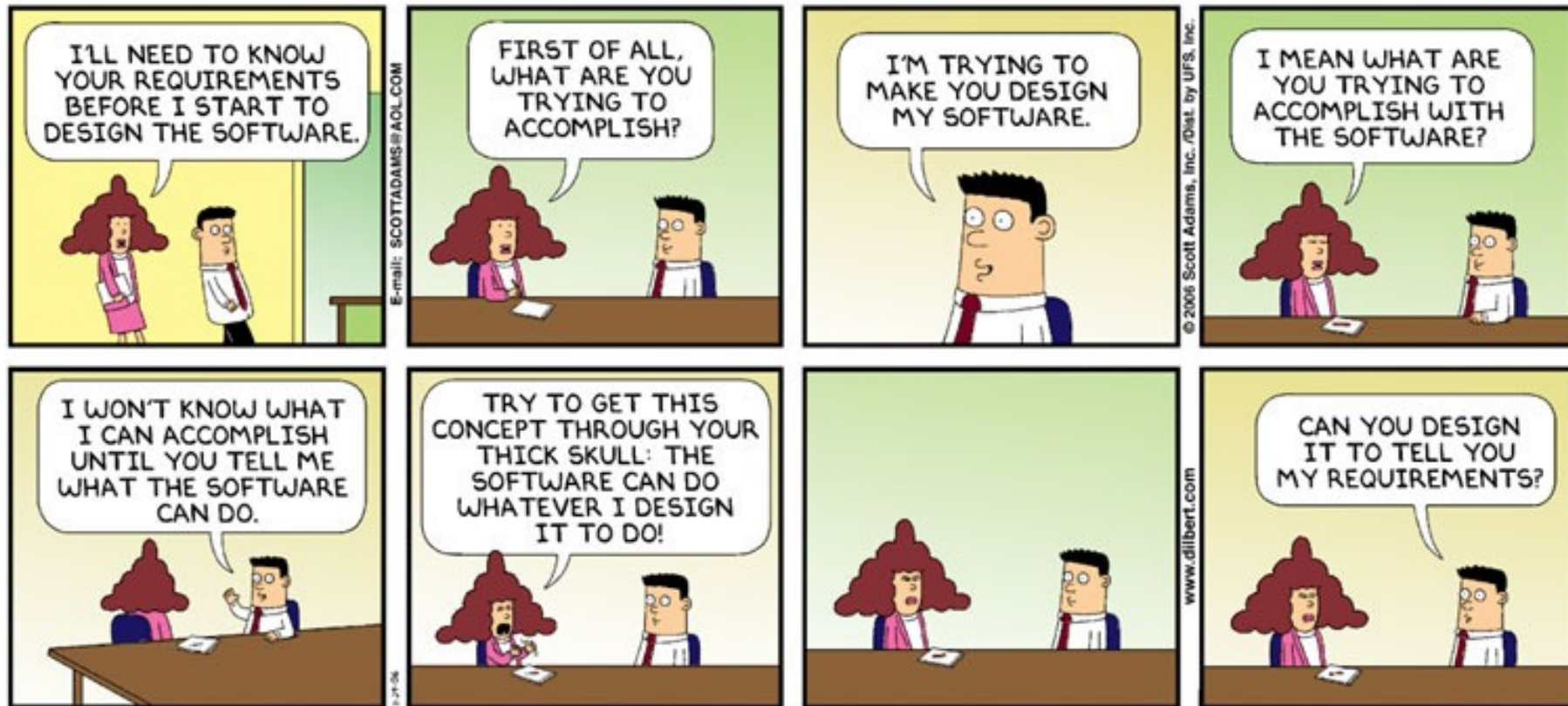
Pushes **incrementality** and **exploration** to the extreme

- **Fixed-length increments** of 1-6 weeks
- **Product owner** or **customer** representative always **available** and has power to make immediate **decisions**
- Only **goals** and **vision** established **upfront**
- Requirements **loosely specified** as **stories**
- Details captured in **test cases**
- At the beginning of each increment
 - **Customer prioritizes** requirements
 - **Developers select** requirements to be implemented in that increment
- **Short feedback cycle** from requirements to deployed system

Characteristics of an “ideal” RE process

- Strongly **interactive**
- **Close** and **intensive collaboration** between
 - Stakeholders (know the domain and the problem)
 - Requirements engineers (know how to specify)
- Very **short feedback** cycles
- **Risk**-aware and **feasibility**-aware
 - Technical risks/feasibility
 - Deadline risks/feasibility
- Careful negotiation / resolution of conflicting requirements
- Focus on establishing **shared understanding**
- Strives for **innovation**

7 Requirements elicitation



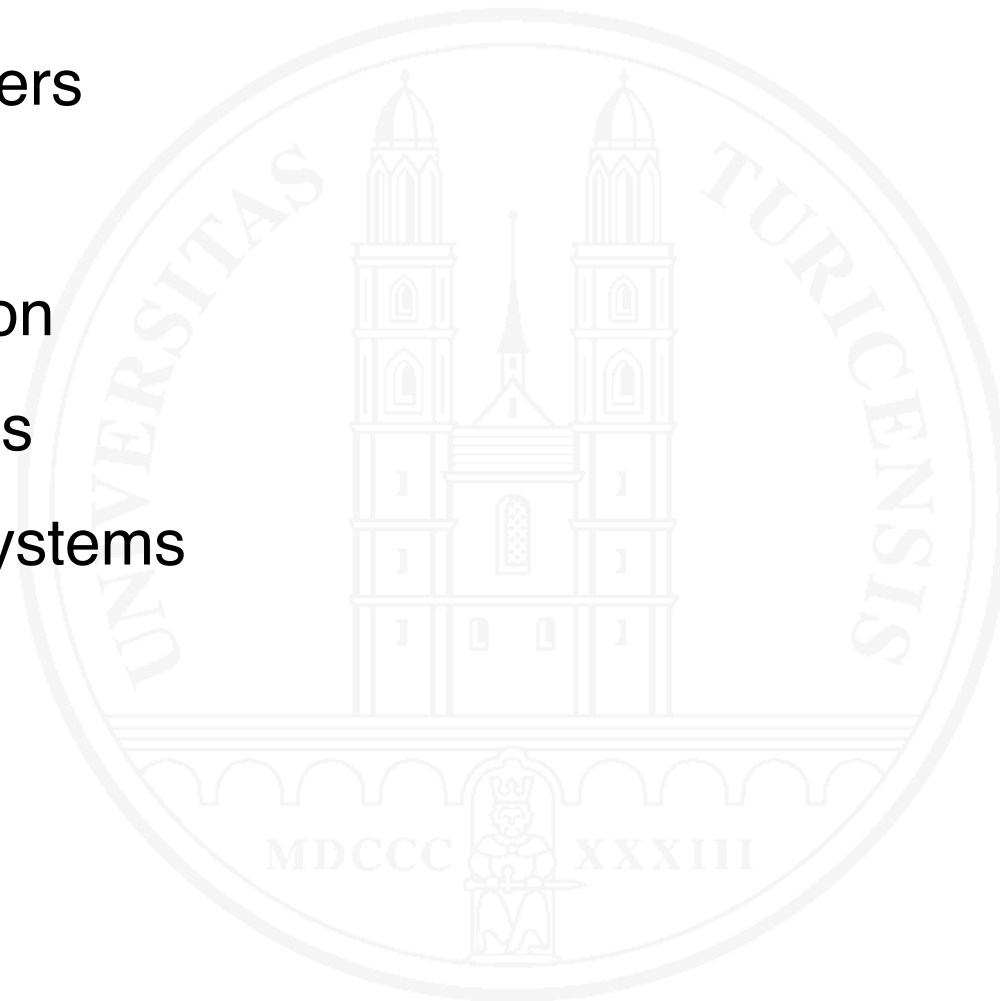
Definition and principles

DEFINITION. **Requirements elicitation** – The process of **seeking, capturing** and **consolidating** requirements from available sources. May include the **re-construction** or **creation** of requirements.

- Determine the stakeholders' **desires** and **needs**
- Elicit information from all available **sources** and **consolidate** it into **well-documented requirements**
- Make stakeholders **happy**, not just satisfy them
- Every elicited and documented requirement must be **validated** and **managed**
- Work **value-oriented** and **risk-driven**

Information sources

- Stakeholders
- Context
- Observation
- Documents
- Existing systems



Stakeholder analysis

Identify stakeholder roles

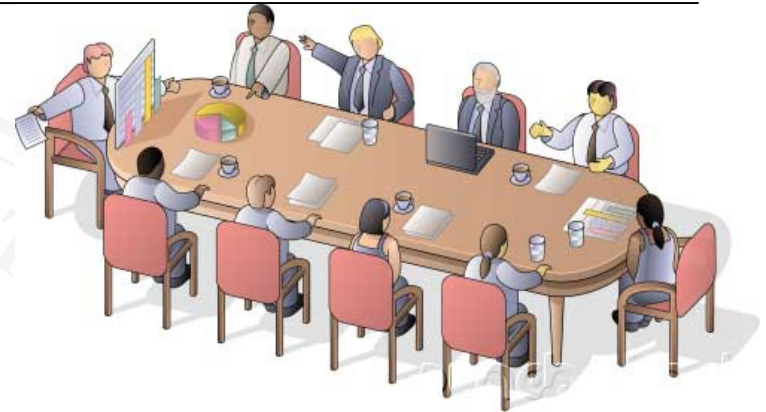
End user, customer, operator,
project manager, regulator,...

In complex cases: Build model of stakeholder goals, dependencies and rationale

Classify stakeholders

- Critical
- Major
- Minor

Identify/determine concrete persons for each stakeholder role



[Yu 1997]

[van Lamsweerde 2001]

[Glinz and Wieringa 2007]

Context analysis

Determine the system's **context** and the context **boundary**

Identify context constraints

- Physical, legal, cultural, environmental
- Embedding, interfaces



Photo © Universitätsklinikum Halle (Saale)

Identify assumptions about the context of your system and make them **explicit**

Map real world phenomena adequately...

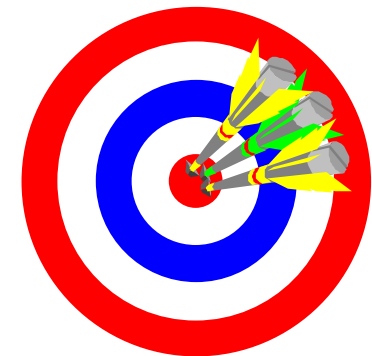
- ... on the required system properties and capabilities
- ... and vice-versa

Goal analysis

Knowing your destination is more important than the details of the timetable.

Before eliciting detailed requirements, the general **goals** and **vision** for the system to be built must be clear

- What are the main goals?
- How do they relate to each other?
- Are there goal conflicts?



Mini-Exercise

Consider the chairlift access control case study.

- (a) Perform a stakeholder analysis.
- (b) How can you map the context property that a skier passes an unlocked turnstile to a system property which can be sensed and controlled by the system?
- (c) Identify some business goals.

Elicitation techniques

Ask

- Interview stakeholders
- Use questionnaires and polls

Collaborate

- Hold requirements workshops

Build and play

- Build, explore and discuss prototypes and mock-ups
- Perform role playing



[Zowghi and Coulin 2005]
[Dieste, Juristo, Shull 2008]
[Gottesdiener 2002]
[Hickey and Davis 2003]
[Goguen and Linde 1993]

Elicitation techniques – 2

Observe

- Observe stakeholders in their work context

Analyze

- Analyze work artifacts
- Analyze problem/bug reports
- Conduct market studies
- Perform benchmarking

Which technique for what?

Technique	Suitability for			
	Express needs	Demonstrate opportunities	Analyze system as is	Explore market potential
Interviews	+	-	+	0
Questionnaires and polls	0	-	+	+
Workshops	+	0	0	-
Prototypes and mock-ups	0	+	-	0
Role play	+	0	0	-
Stakeholder observation	0	-	+	0
Artifact analysis	0	-	+	-
Problem/bug report analysis	+	-	-	0
Market studies	-	-	0	+
Benchmarking	0	+	-	+

Typical problems

Inconsistencies among stakeholders in

- needs and expectations
- terminology

Stakeholders who know their needs, but **can't express** them

Stakeholders who **don't know** their needs

Stakeholders with a **hidden agenda**

Stakeholders thinking in **solutions** instead of problems

Stakeholders frequently **neglect attributes** and **constraints**

↳ Elicit them explicitly

Who should elicit requirements?

- Stakeholders must be involved
- Domain knowledge is essential
 - Stakeholders need to have it (of course)
 - Requirements engineers need to know the main domain concepts
 - A “smart ignoramus” can be helpful [Berry 2002, Sect. 7]
- Don’t let stakeholders specify themselves without professional support
- Best results are achieved when stakeholders and requirements engineers collaborate

Eliciting functional requirements

- Who wants to achieve what with the system?
- For every identified function
 - What's the desired result and who needs it?
 - Which transformations and which inputs are needed?
 - In which state(s) shall this function be available?
 - Is this function dependent on other functions?
- For every identified behavior
 - In which state(s) shall the system have this behavior?
 - Which event(s) lead(s) to this behavior?
 - Which event(s) terminate(s) this behavior?
 - Which functions are involved?

Eliciting functional requirements – 2

- For every identified **data** item
 - What are the required **structure** and the **properties** of this item?
 - Is it **static** data or a data **flow**?
 - If it's static, must the system keep it **persistently**?
- Analyze **mappings**
 - How do real world functions/behavior/data map to system functions/behavior/data and vice-versa?
- Specify **normal and exceptional** cases

Eliciting quality requirements

Stakeholders frequently state quality requirements in qualitative form:

“The system shall be fast.”

“We need a secure system.”

Problem: Such requirements are

- Ambiguous
- Difficult to achieve and verify
- Classic approach:
 - Quantification → ⊕ measurable ⊖ maybe too expensive
 - Operationalization → ⊕ testable ⊖ implies premature design decisions

New approach to eliciting quality requirements

[Glinz 2008]

Represent quality requirements such that they deliver **optimum value**

Value of a requirement = **benefit** of development risk reduction **minus cost** for its specification

- Assess the criticality of a quality requirement
- Represent it accordingly
- Broad range of possible representations

The range of adequate representations

Situation	Representation	Verification
1. Implicit shared understanding	Omission	Implicit
2. Need to state general direction Customer trusts supplier	Qualitative	Inspection
3. Sufficient shared understanding to generalize from examples	By example	Inspection, (Measurement)
4. High risk of not meeting stake- holders' desires and needs	Quantitative in full	Measurement
5. Somewhere between 2 and 4	Qualitative with partial quantification	Inspection, partial measurement

Eliciting performance requirements

Things to elicit

- **Time** for performing a task or producing a reaction
- **Volume** of data
- **Throughput** (data transmission rates, transaction rates)
- **Frequency** of usage of a function
- **Resource consumption** (CPU, storage, bandwidth, battery)
- **Accuracy** (of computation)

Eliciting performance requirements – 2

- What's the meaning of a performance value:
 - Minimum?
 - Maximum?
 - On average?
 - Within a given interval?
 - According to some probability distribution?
- How much deviation can be tolerated?

Eliciting specific quality requirements

- Ask stakeholders explicitly
- A quality model such as ISO/IEC 25010:2011 (formerly ISO/IEC 9126) can be used as a checklist
- Quality models also help when a specific quality requirement needs to be quantified

Eliciting constraints

- Ask about **restrictions** of the potential **solution space**
 - **Technical**, e.g., given interfaces to neighboring systems
 - **Legal**, e.g., restrictions imposed by law, standards or regulations
 - **Organizational**, e.g. organizational structures or processes that must not be changed by the system
 - **Cultural, environmental, ...**
- Check if a requirement is **concealed** behind a constraint
 - Constraint stated by a stakeholder: **“When in exploration mode, the print button must be grey.”**
 - Actual requirement: **“When the system is used without a valid license, the system shall disable printing.”**

Mini-Exercise

Consider the chairlift access control case study.

- (a) Which technique(s) would you select to elicit requirements from the chairlift ticket office clerks?
- (b) How, for example, can you achieve consensus among the ski resort management, the technical director of chairlifts, the ticket office clerks, and the service employees?
- (c) Identify some constraints for the chairlift access control system.

Analysis of elicited information

Structure-oriented

Analyze terminology /
domain properties
Build glossary

Analyze business
and data objects
Build object and
class models

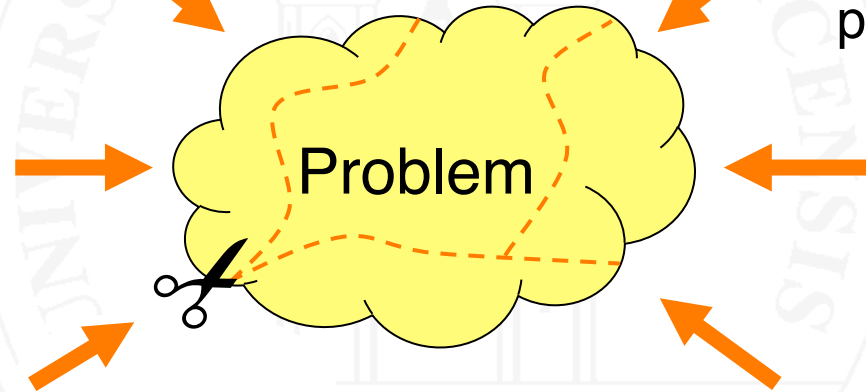
Decompose problem
Build hierarchical structure

Process-oriented

Analyze processes /
workflows
Build activity /
process models

Analyze dynamic
system behavior
Build behavior
model

Analyze actor-system interaction
Build scenarios / use cases



Note: requirements are about a future state of affairs; analyze the current state only when necessary

Documenting elicited requirements

Build specification **incrementally** and **continuously**

Document requirements in **small units**

End over means: Result → Function → Input

Consider the **unexpected**: specify non-normal cases

Quantify critical attributes

Document critical **assumptions explicitly**

Avoid **redundancy**

Build a **glossary** and stick to terminology defined in the glossary

8 Specifying with natural language

The system shall ...

The oldest...

...and most widely used way

- taught at school
- extremely expressive

But not the best

- Ambiguous
- Imprecise
- Error-prone
- Verification only by careful reading



Michelangelo's Moses (San Pietro in Vincoli, Rome)
Moses holds the Ten Commandments
in his hand: written in natural language

Problems with natural language requirements

Read the subsequent requirements. Any findings?

“For every turnstile, the total number of turns shall be read and archived once per day.”

“The system shall produce lift usage statistics.”

“Never shall an unauthorized skier pass a turnstile.”

“By using RFID technology, ticket validation shall become faster.”

“In the sales transaction, the system shall record the buyer’s data and timestamp the sold access card.”

Some rules for specifying in natural language

[Rupp et al. 2009]

- Use **active voice** and defined subjects
- Build phrases with **complete** verbal structure
- Use terms as defined in the **glossary**
- Define precise meanings for **auxiliary verbs** (shall, should, must, may,...) as well as for process verbs (for example, “produce”, “generate”, “create”)
- Check for nouns with **unspecific semantics** (“the data”, “the customer”, “the display”,...) and replace where appropriate
- When using adjectives in comparative form, specify a **reference point**: “better” → “better than”

More rules

- Scrutinize **all-quantifications**: “every”, “always”, “never”, etc. seldom hold without any exceptions
- Scrutinize **nominalizations** (“authentication”, “termination”...): they may conceal incomplete process specifications
- State **every requirement** in a **main clause**. Use subordinate clauses only for making the requirement more precise
- Attach a **unique identifier** to every requirement
- **Structure** natural language requirements by ordering them in **sections** and **sub-sections**
- Avoid **redundancy** where possible: “never ever” → “never”

Phrase templates

[Rupp et al. 2009
ISO/IEC/IEEE 29148:2011]

Use **templates** for creating **well-formed** natural language requirements

Typical template:

[<Condition>] <Subject> <Action> <Objects> [<Restriction>]

Example:

When a valid card is sensed, the system shall send
the command 'unlock_for_a_single_turn' to the turnstile
within 100 ms.

Agile stories

[Cohn 2004]

- A **single sentence** about a requirement
- Written from a **stakeholder's perspective**
- Optionally including the **expected benefit**
- Accompanied by **acceptance criteria** for requirement
- Acceptance criteria make the story more precise

Standard **template**:

As a **<role>** I want to **<my requirement>** [so that **<benefit>**]

A sample story

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

Sample acceptance criteria

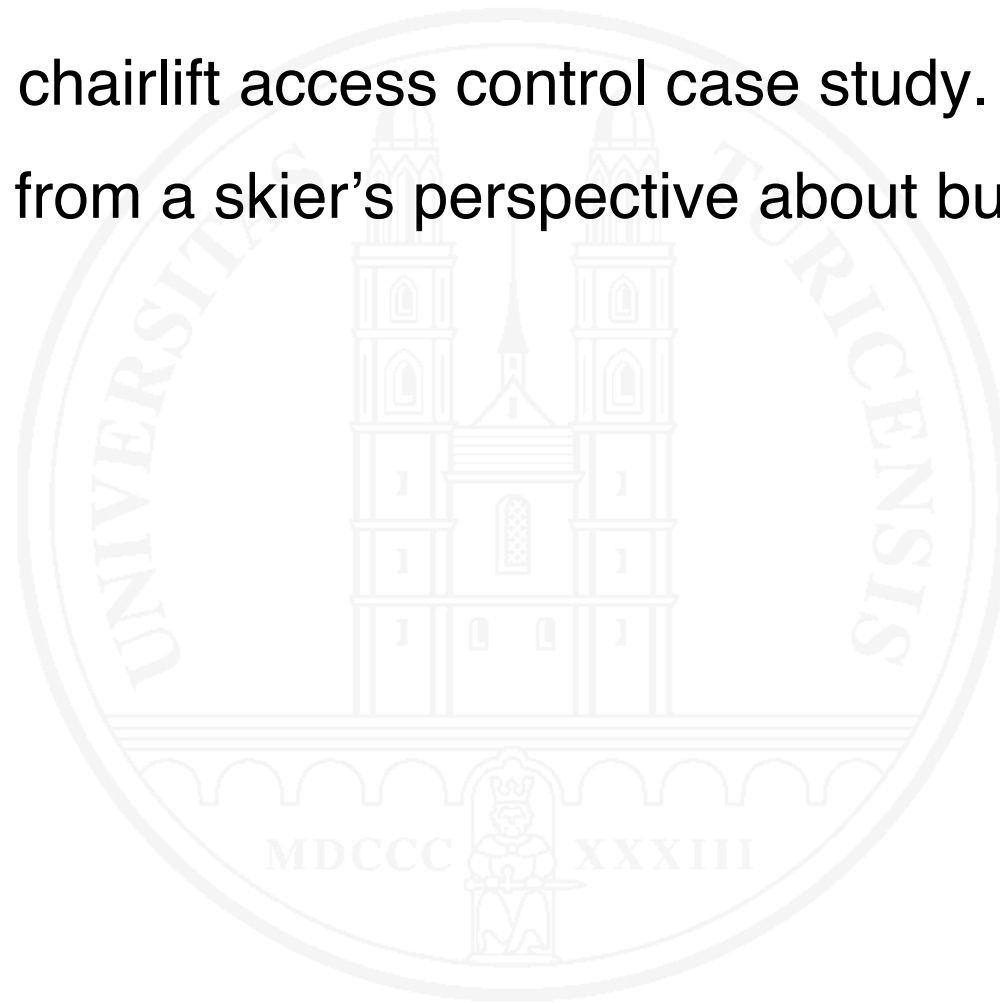
Acceptance criteria:

- 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

Mini-Exercise: Writing a user story

Consider the chairlift access control case study.

Write a story from a skier's perspective about buying a day card.



All-quantification and exclusion

- Specifications in natural language frequently use all-quantifying or excluding statements without much reflection:

“When operating the coffee vending machine, the user shall **always** be able to terminate the running transaction by pressing the cancel key.”

Also when the coffee is already being brewed or dispensed?

- ⇒ **Scrutinize all-quantifications** (“every”, “all”, “always”...) and **exclusions** (“never”, “nobody”, “either – or”,...) **for potential exceptions**
- ⇒ **Specify found exceptions** as requirements

Dealing with redundancy

- Natural language is frequently (and deliberately) **redundant**
 - Secures **communication success** in case of some information loss
- In requirements specifications, redundancy is a **problem**
 - Requirements are specified **more than once**
 - In case of modifications, all redundant information must be **changed consistently**
- Make redundant statements only when needed **for abstraction purposes**
- Avoid **local redundancy**: “never ever” → “never”