Part I: The Fundamentals

Part II: Requirements Engineering Practices

Part III: Enablers and Stumbling Blocks

Conclusions

References

5 Documenting requirements

Bridging the gap:

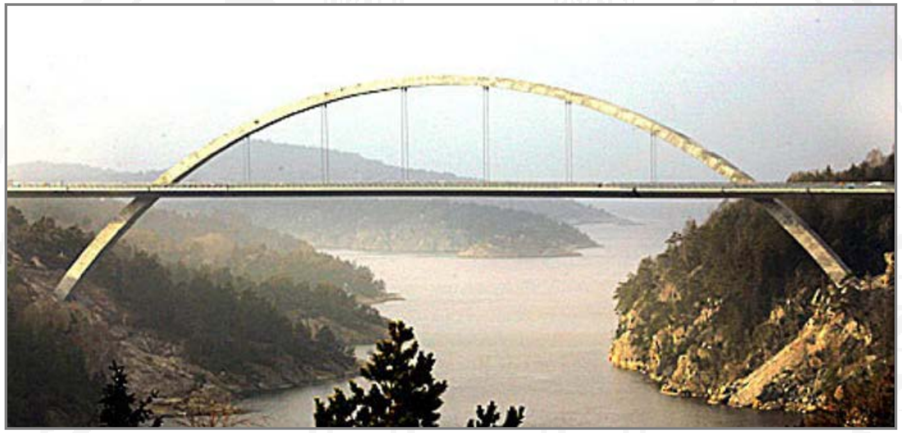
Stakeholders  System builders

Photo © Lise Aserud / DPA

The need:

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

The means: Documented requirements

5.1 Requirements Engineering work products

DEFINITION. Work product – A recorded, intermediate or final result of information generated in a work process.
Synonym: artifact

Work products are characterized by their

- Purpose
- Representation (free text, structured text, lists, graphics, drawings,...)
- Size (single requirements, sets of requirements, documents (or document-like structures)
- Lifespan (temporary, evolving, durable)

Note that a work product may contain other work products

Work products and their purposes

Single requirements

- Sentence in natural language – expressing an individual requirement
- User story – specifying a function or behavior from a stakeholder's perspective

Work products and their purposes – 2

Sets of requirements

- Use case – specifying a system function from a stakeholder's perspective
- Graphic model – specifying various aspects, e.g., context, activity, behavior
- Task description – specifying a task to perform
- External interface – specifying the information exchanged between a system and an actor in the system context
- Epic – providing a high-level view of a stakeholder need
- Feature – A distinguishing characteristic of a system that provides value for stakeholders

Work products and their purposes – 3

Documents and document-like structures

- System requirements specification, business requirements specification, stakeholder or user requirements specification – providing a baselined or released requirements document
- Product and sprint backlog – managing a list of work items, including requirements
- Story map – visual arrangement of user stories
- Vision – a conceptual imagination of a future system or product

Work products and their purposes – 4

Other RE-related work products

- Glossary – providing an unambiguous and agreed common terminology
- Textual note or graphic sketch – serving for communication and understanding
- Prototype – understanding or validating requirements

5.2 Classic requirements specifications

Full-fledged requirements specifications are typically needed

- When customers want contractually fixed requirements, costs and deadlines
- When systems are built by an external contractor based on a set of given requirements (tendering, outsourcing)
- In regulated environments where regulators check compliance of developed systems to their requirements

Document types

[ISO/IEC/IEEE 2018]

- Stakeholder requirements specification (also called customer requirements specification)
  What the stakeholders want (independent of any system providing it)
- System requirements specification
  The system or product to be developed and its context
- Software requirements specification
  If the system is a pure software system
- Business requirements specification
  High-level specification of business needs or goals

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

5.3 Requirements in agile development

No classic requirements specification document (unless mandated by regulators)

Various work products that ...

- ... specify requirements: vision, stories, epics, use cases,...
- ... have requirements-related content: Prototypes, mock-ups, storyboards, roadmap, early product versions (e.g., MVP – minimum viable product)

Value-driven creation of work products

Agile requirements work products

- Requirements primarily captured as a collection of user stories, organized in a product backlog
- A system vision provides an abstract overview of the system to be developed
- On an intermediate level of abstraction, epics and features can serve to group user stories
- Stories may be sub-divided into tasks
- Use cases/scenarios and other models may be used to provide structure and context

5.4 Glossary

RE typically is a multi-person endeavor

→ Danger of missing shared understanding in terminology

DEFINITION. Glossary – A collection of definitions of terms that are relevant in some domain.

A glossary defines

- Context-specific terms
- Everyday terms that have a special meaning in the given context
- Abbreviations and acronyms

Rules for creating and maintaining a glossary

- Consistently structured
- Centrally managed
- Defined responsibilities for creation and maintenance
- Maintained over the entire course of a project
- Usage of terms as defined in the glossary is mandatory
- Stakeholders must agree upon the glossary
- Synonyms and Homonyms properly treated
  - Synonyms (different terms denoting the same thing) marked
  - Homonyms (same term for different things) avoided or marked

5.5 Prototypes

DEFINITION. Prototype – In software and systems engineering: A preliminary, partial realization of certain characteristics of a system.

Serves for exploring, communicating or validating concepts and requirements.

The realization may be in any physical form, from paper and sticky notes over clickable pages to executable source code.

In RE, a prototype is a means for

- specifying requirements by example
- validating requirements
- supporting stakeholder communication and shared understanding

Forms of Prototypes in RE

[Lichter et al. 1994]

- *Exploratory prototype*:
  - Creating shared understanding
  - Clarifying requirements
  - Validating requirements on different levels of fidelity
  - Thrown away after use

- *Evolutionary prototype*:
  - Pilot system forming the nucleus of a system to be developed
  - Final system evolves by incrementally extending and improving the prototype

Exploratory prototypes

- *Wireframe*
  - Low-fidelity prototype
  - Built with paper or other simple materials
  - Primarily serves for discussing and validating design ideas and user interface concepts
- *Mock-up*
  - Medium-fidelity prototype
  - Demonstrates characteristics of a user interface without implementing any real functionality
  - Real screens and click flows, but without functionality behind
  - Primarily serves for specifying and validating user interfaces

Exploratory prototypes – 2

- *Native prototype*
  - High-fidelity prototype
  - Implements critical parts of a system to an extent that stakeholders can work with the prototype
  - Primarily serves for validating that the prototyped part of the system will work and behave as expected

Exploratory prototypes can be expensive work products

- Choose proper level of fidelity
- Trade-off between cost and value gained

5.6 Aspects to be documented

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

- Functionality
  - Structure and Data: Static structure, (persistent) data
  - Function and Flow: Functions (results, preconditions, processing), flow of control and data
  - State and Behavior: State-dependent dynamic system behavior as observable by users
  - Both normal and abnormal cases must be specified

Aspects to be documented – 2

- Quality

  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, Availability, etc.

Aspects to be documented – 3

- Constraints

  Restrictions that must be obeyed / satisfied

  - Technical: given interfaces or protocols, etc.
  - Legal: laws, standards, regulations
  - Organizational: given structures, policies, processes
  - Cultural: culturally shaped user habits and expectations
  - Environmental: e.g., energy consumption, heat dissipation
  - Physical: laws of physics, properties of materials
  - Solutions / restrictions demanded by important stakeholders

Aspects to be documented – 4

- Context and Boundary
  - Domain requirements and domain assumptions in the context of a system
  - Embedding of a system in its context
  - Interaction between a system and the actors in the context

5.7 How to document

Sample standards for classic requirements documents

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

VOLERE

- 27 chapters
- System and project requirements

IREB

- Simple template for system/software requirements specifications

Enterprise-specific standards

- Imposed by customer or given by supplier

IEEE Std 830-1998

[IEEE 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

VOLERE

[Robertson and Robertson 2012]
[https://www.volere.org/templates/]

Project Drivers
1. The Purpose of the Project
2. The Stakeholders

Project Constraints
3. Mandated Constraints
4. Naming Conventions and Terminology
5. Relevant Facts and Assumptions

Context and Functionality
6. The Scope of the Work
7. Business Data Model & Data Dictionary
8. The Scope of the Product
9. Functional Requirements

Non-Functional Requirements
10. Look and Feel Requirements
11. Usability and Humanity Requirements
12. Performance Requirements
13. Operational & Environmental Requirements
14. Maintainability and Support Requirements
15. Security Requirements
16. Cultural Requirements
17. Compliance Requirements

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

Subtitles added by MG, inspired by an earlier version of the template

A simple document template

[Glinz et al. 2020]

Part I: Introduction

1. System purpose
2. Scope of system development
3. Stakeholders

Part II: System Overview

4. System vision and goals
5. System context and boundary
6. Overall system structure
7. User Characteristics

Part III: System requirements

Organized hierarchically according to system structure

Per sub-system/component:

- Functional requirements (structure and data – function and flow – state and behavior)
- Quality requirements
- Constraints
- Interfaces

References

Appendices

- Glossary
- Assumptions and dependencies

Guidelines for agile requirements

- Standard template for writing user stories (cf. Chapter 8)
- Organizing stories in a product backlog
- Artifact / work product structures provided by textbooks

[Leffingwell 2011]

General guideline: do things only if they add value

How to document – language options

Informally

- Plain natural language (narrative text)

Semi-formally

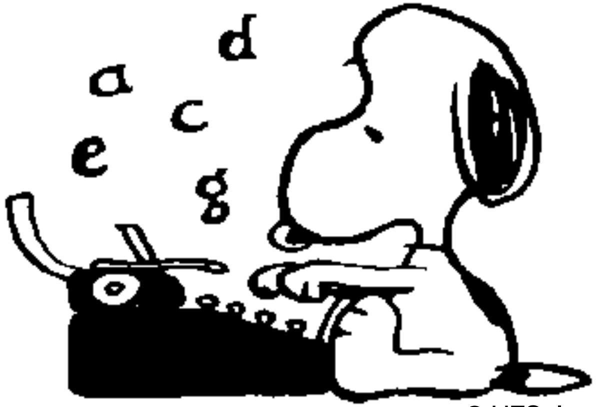
- Structured natural language (using templates or forms)
- Graphic models — Typically as diagrams which are enriched with natural language text

Formally

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

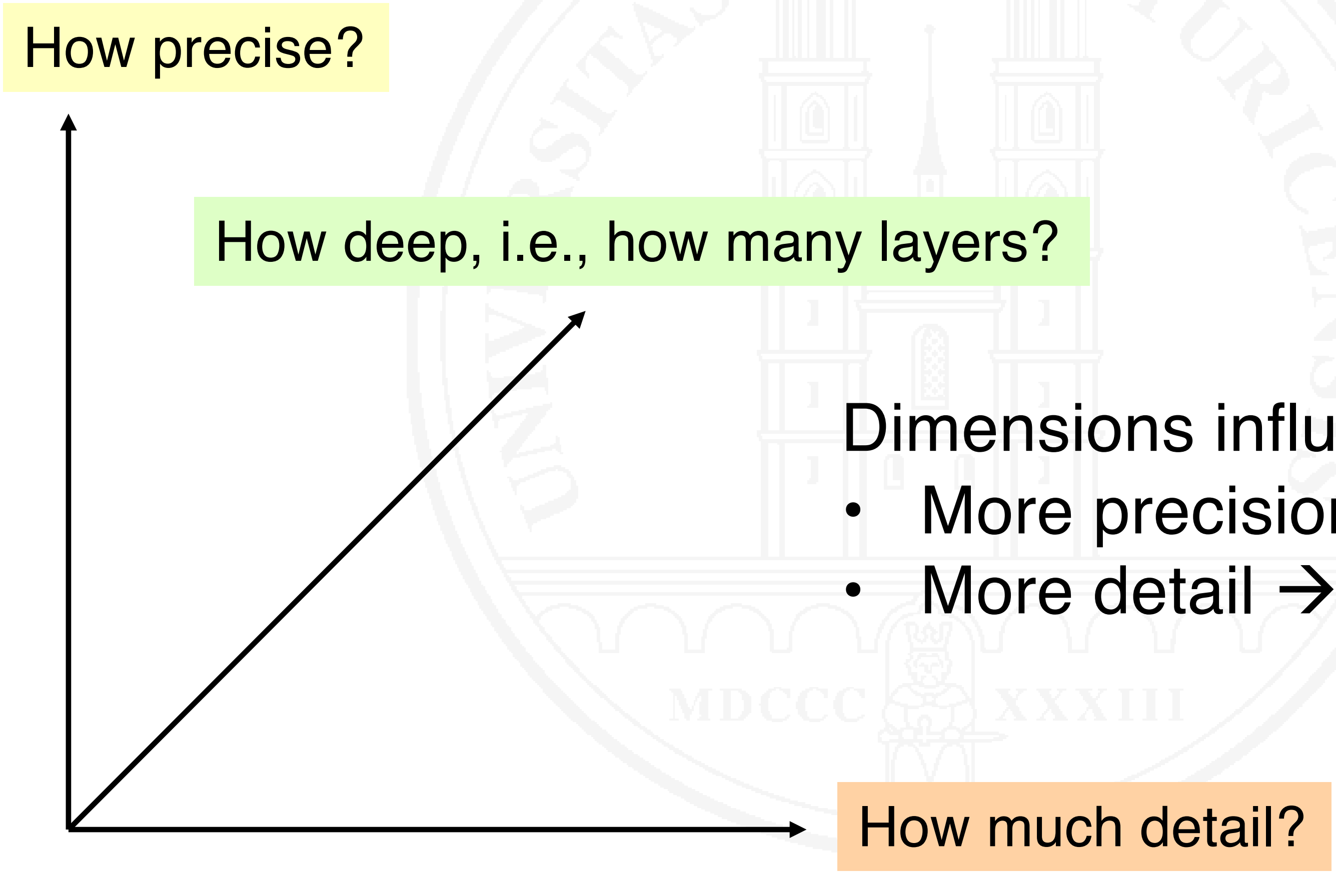
General rules for requirements documentation

- Specify requirements as small, identifiable units whenever possible
- Record metadata such as source, author, date, status
- Use structure templates
- Adapt the degree of detail to the risk associated with a requirement
- Specify normal and exceptional cases
- Don't forget quality requirements and constraints



Precision – Detail – Depth

Three dimensions:



Dimensions influence each other:

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

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

..."

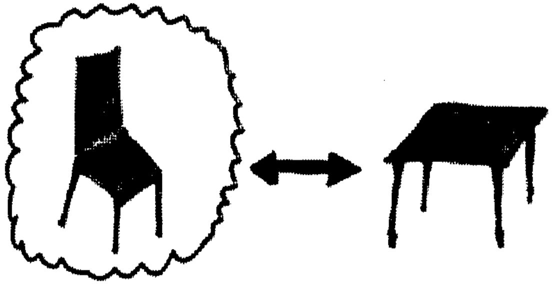
5.8 Quality of documented requirements

Two aspects of requirements quality

- Quality of individual requirements
- Quality of requirements work products, for example, a requirements specification

Hint: Don't confuse quality of requirements with quality requirements

Quality of individual requirements



For individual requirements, strive for requirements that are...

- Adequate — True and agreed stakeholder needs
- Understandable — Prerequisite for shared understanding
- Verifiable — Conformance of implementation can be checked
- Unambiguous — True shared understanding
- Complete — No missing parts
- Necessary — Part of the relevant system scope
- Feasible — Non-feasible requirements are a waste of effort

Quality of requirements work products

When creating a requirements work product, strive for a work product that is

• Consistent	No contradictions
• Complete	Contains the relevant requirements
• Conformant	Conforms to prescribed work product structure, format or style
• Modifiable	Because change will happen
• Non-redundant	Requirements do not overlap
• Structured	Improves readability of work product
• Traceable	Linked to related artifacts

Quality criteria are in the eye of the beholder

- No general consensus
- Different, overlapping sets of quality criteria used in
  - this course
  - RE textbooks
  - RE standards (e.g., ISO/IEC/IEEE 29148:2018)
  - Quasi-standards such as the IREB Certified Professional for Requirements Engineering (see http://www.ireb.org)

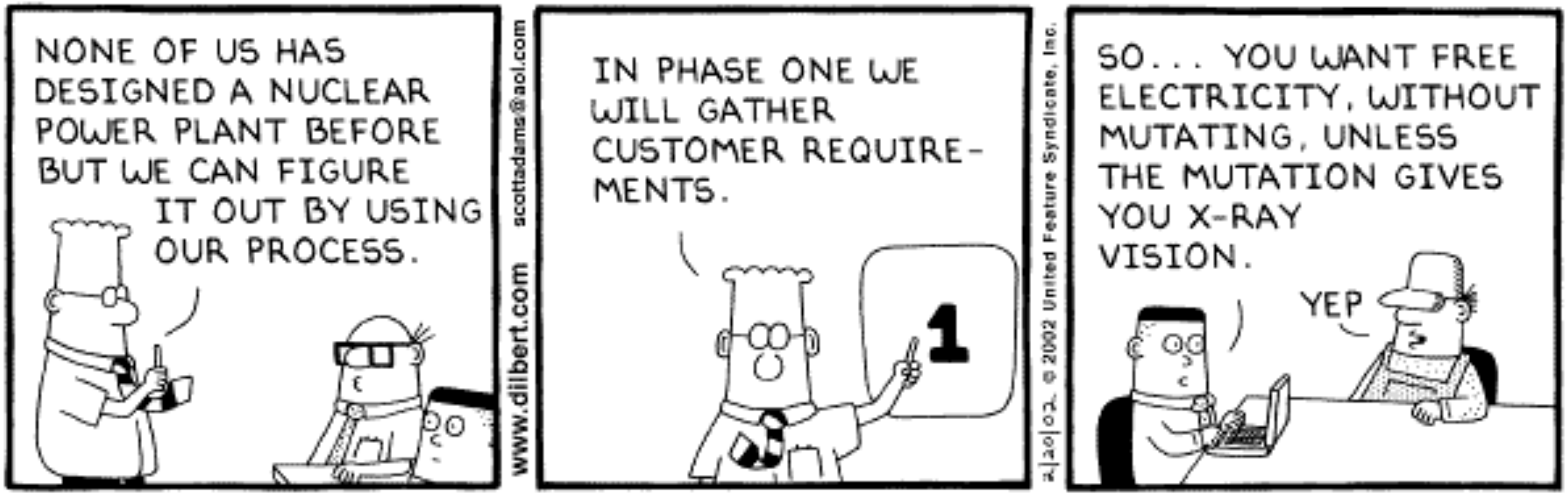
Not all qualities are equally important

- Adequacy and understandability are key
- Verifiability and Consistency are very important
- Achieving total completeness and unambiguity is neither possible nor economically feasible in most cases
- The importance of feasibility, traceability, conformance, etc. of requirements depends on the concrete project/situation

☞ Strive for value, not for blind satisfaction of requirements quality criteria!

6 Requirements Engineering processes

DEFINITION. Process – A set of interrelated activities performed in a given order to process information or materials.



[Armour 2004, Reinertsen 1997, 2009]

The principal tasks

Requirements Specification

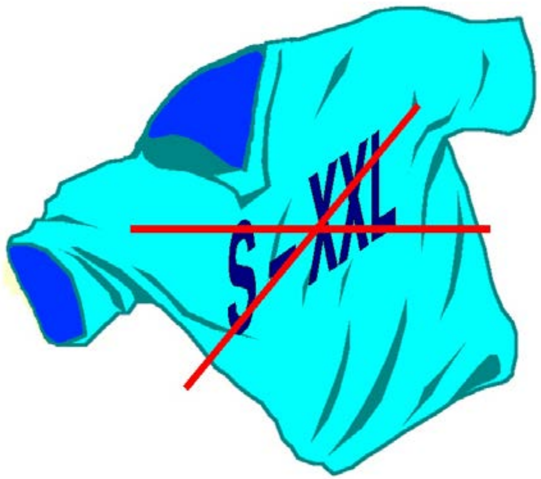
- Elicitation & Analysis
- Documentation
- Validation

Requirements Management

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

An RE process organizes how to carry out RE tasks, using appropriate practices and producing needed work products

No ‘one size fits all’ process



Some influencing factors

- Overall process fit
- Development context
- Stakeholder availability and capability
- Shared understanding
- Complexity and criticality
- Constraints
- Time and budget available
- Volatility of requirements
- Experience of requirements engineers

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

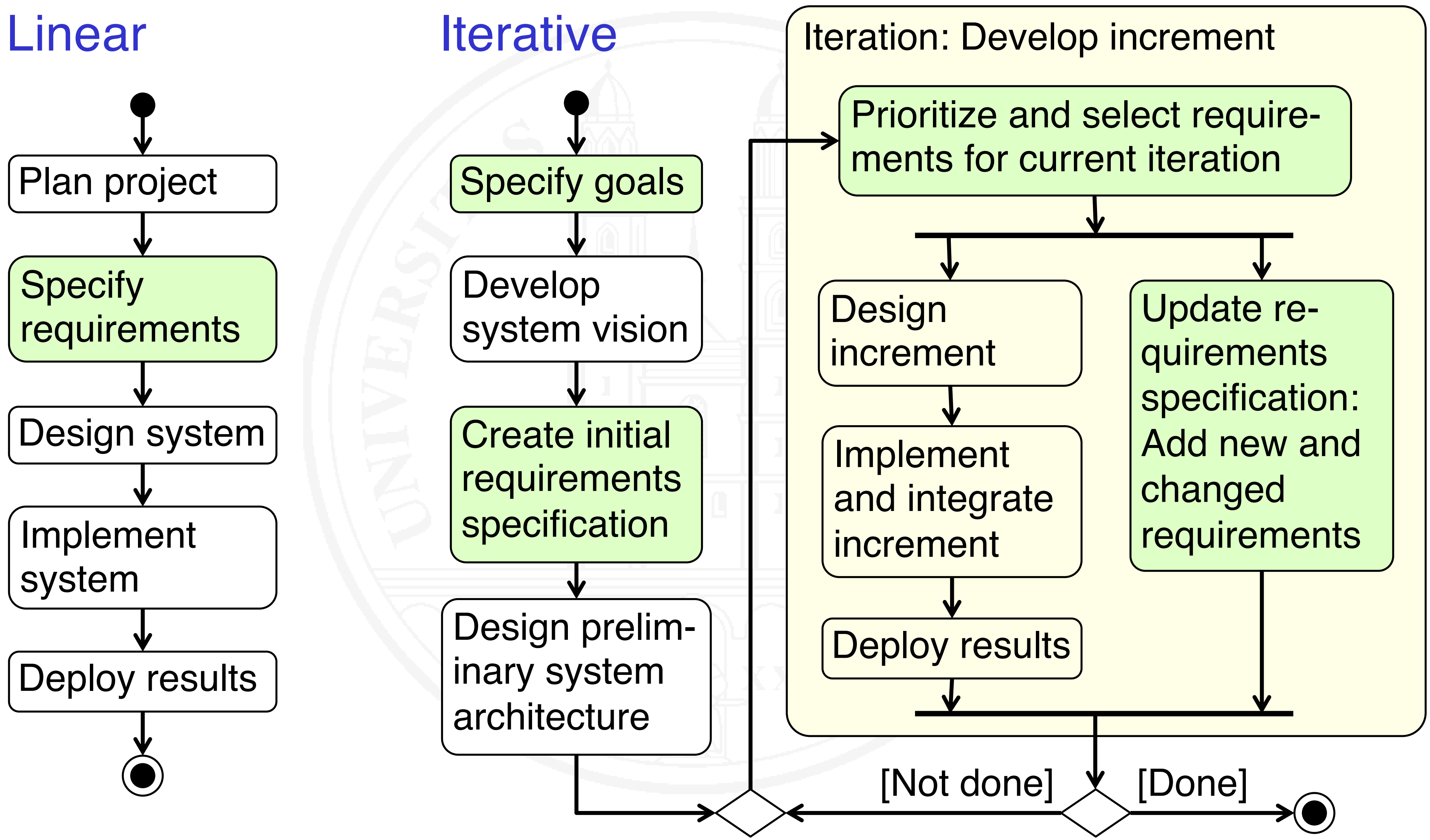
Process facets

There are three process facets, from which an RE process can be configured

- Time facet: Linear vs. Iterative
- Purpose facet: Prescriptive vs. Explorative vs. COTS-Driven
- Target facet: Customer-Specific vs. Market-Oriented

- Selection criteria indicate how to configure the process in each facet

Time facet: Process structure



Time facet: Linear

Requirements are specified up front in a single phase of the process

Selection criteria:

- System development process is plan-driven and mostly linear
- Stakeholders can specify their requirements up front
- Comprehensive requirements specification required as a contractual basis for outsourcing design and implementation
- Regulatory authorities require a requirements specification

Time facet: Iterative

Requirements are specified incrementally, starting with general goals and then adding or modifying requirements in every iteration

Selection criteria:

- System development process is iterative and agile
- Evolving requirements – not known up front
- Stakeholders are available such that short feedback loops established for mitigating risk
- Duration of project allows for more than 1-2 iterations
- Ability to change requirements easily is important

Purpose facet: Prescriptive

Requirements specification is a contract: All requirements are binding and must be implemented

Selection criteria:

- Customer requires fixed-price contract
- Functionality determines cost and deadlines
- Design and implementation tendered or outsourced

Purpose facet: Explorative

Only goals known, concrete requirements have to be explored

Selection criteria:

- Stakeholders only have a vague idea about their requirements
- Stakeholders strongly involved, provide continuous feedback
- Deadlines and cost take precedence over functionality
- Customer is satisfied with a framework contract
- Not a priori clear which requirements actually shall be implemented and in which order → Prioritization needed

Purpose facet: COTS-Driven

COTS-Driven

Requirements must reflect functionality of chosen COTS solution

Selection Criteria:

- System will be implemented with COTS software
- Only requirements not covered by the COTS solution shall be specified

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

Target facet: Customer-Specific

System is ordered by a customer and developed by a supplier for this customer

Selection criteria:

- The system will be mainly used by the organization that has ordered the system and pays for its development.
- The important stakeholders are mainly associated with the customer's organization.
- Individual persons can be identified for the stakeholder roles.
- The customer wants a requirements specification that can serve as a contract.

Target facet: Market-Oriented

System is developed as a product or service for a market

Selection criteria:

- Developing organization (or one of its clients) intends to sell the system as a product or service in some market segment
- Prospective users not individually identifiable
- Requirements engineers have to design the requirements so that they match the envisaged needs of the targeted users
- Product owners, marketing people, digital designers and system architects are primary stakeholders

Hints and caveats

- Linear RE processes only work if a sophisticated process for changing requirements is in place
- Linear RE processes imply long feedback loops: intensive validation of requirements must be performed
- Market-oriented RE processes crucially depend on fast feedback from pilot users for validating whether the product will actually satisfy needs of the targeted user segment
- In an agile setting, an iterative and explorative RE process fits best

Facet combinations

- *Linear* and *prescriptive* are frequently chosen together
- *Explorative* processes are typically also *iterative*
- *Market-Oriented* does not combine well with *Linear* and *Prescriptive*

How to configure an RE process

1 Analyze the influencing factors

2 Assess the facet criteria

3 Configure
- Select one of the subsequent typical configurations where appropriate
- Otherwise choose what is most appropriate with respect to value and risk

4 Determine main work products to be produced

5 Select appropriate practices for the tasks to be performed according to the chosen process

Typical RE process configurations

Participatory: Iterative & Explorative & Customer-Specific

- Main application case
  Supplier and customer closely collaborate; customer stakeholders strongly involved both in specification and development processes
- Typical work products
  Product backlog with user stories and/or task descriptions, vision, prototypes
- Typical information flow
  Continuous interaction between stakeholders, product owners, requirements engineers, and developers

Typical RE process configurations – 2

Contractual: Typically Linear (sometimes Iterative) & 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 work products
  Classic system requirements specification, consisting of textual requirements and models.
- Typical information flow
  Primarily from stakeholders to requirements engineers

Typical RE process configurations – 3

Product-oriented: Iterative & Explorative & Market-Oriented

- Main application case
  An organization specifies and develops software in order to sell/distribute it as a product or service
- Typical work products
  Product backlog with user stories and/or task descriptions, vision, prototypes, user feedback
- Typical information flow
  Interaction between product owner, marketing, requirements engineers, digital designers, and developers plus feedback from customers/users

Typical RE process configurations – 4

COTS-Aware: [Iterative I 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
- Typical work products:
  Process models describing the alignment of business processes and the COTS solution, partial requirements specification, covering what is not provided by the COTS solution
- Typical information flow:
  Primarily from stakeholders and COTS solution experts to requirements engineers

Agile requirements process

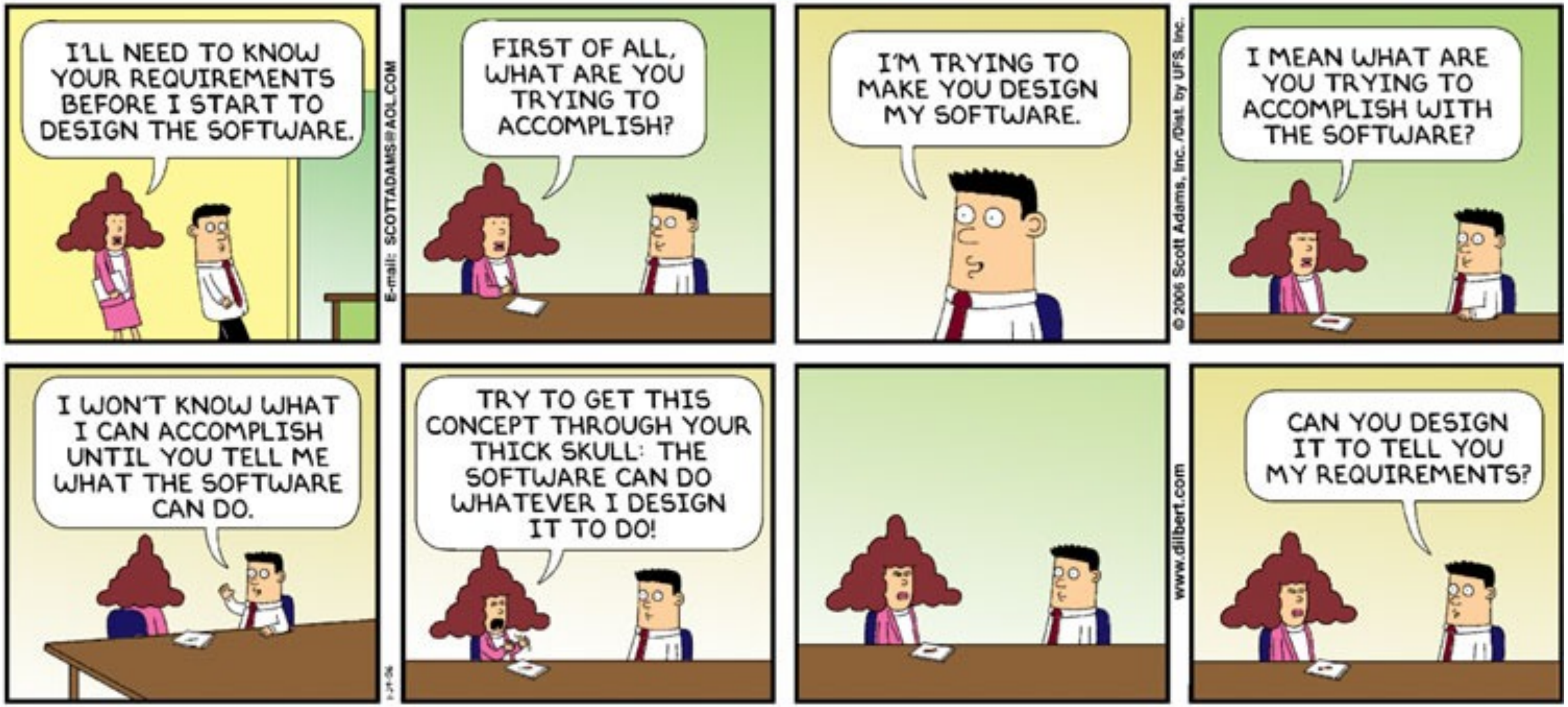
Pushes incrementality and exploration to the extreme

- Fixed-length iterations 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 (with details captured in acceptance criteria)
- Use cases or other means used for providing structure & context
- At the beginning of each iteration
  - Customer/product owner prioritizes requirements
  - Developers select what to implement in that iteration
- 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

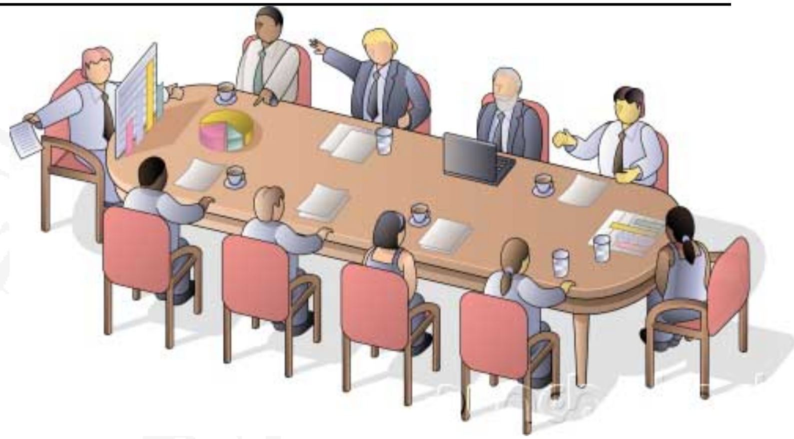
DEFINITION. Requirements elicitation – The process of seeking, capturing and consolidating requirements from available sources, potentially including the re-construction or creation of requirements.

- 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 stake-holder goals, dependencies and rationale

[Yu 1997]
[van Lamsweerde 2001]

Classify stakeholders

[Glinz and Wieringa 2007]

- Critical
- Major
- Minor

Identify/determine concrete persons for each stakeholder role

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)

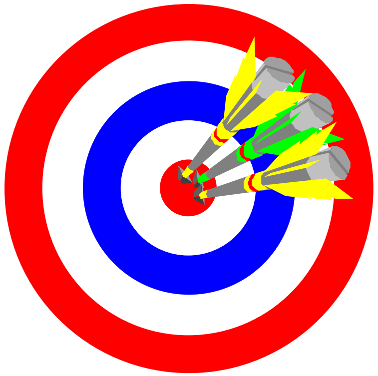
Determine the system scope (cf. Chapter 2.4)

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
- Reply/follow-up to user feedback

Collaborate

- Hold requirements workshops
- Provide community platforms

Build and play

- Build, explore and discuss prototypes (cf. Chapter 5.5)
- Perform role playing

[Zowghi and Coulin 2005]
[Dieste, Juristo, Shull 2008]
[Gottesdiener 2002]
[Hickey and Davis 2003]
Kolpondinos and Glinz 2019]
[Goguen and Linde 1993]

Elicitation techniques – 2

Observe

- Observe stakeholders in their work context

Analyze

- Analyze work products
- Analyze user feedback
  - Direct feedback: problem/bug reports, app reviews, tweets, explicit feedback channels, ...
  - Indirect feedback: user forums, system usage monitoring, ...
- Conduct market studies
- Perform benchmarking

Which technique for what?

Technique	Suitability for			
	Express needs	Demonstrate opportunities	Analyze system as is	Explore market potential
Interviews	+	–	+	o
Questionnaires and polls	o	–	+	+
Workshops, Community platforms	+	o	o	o
Explorative prototypes	o	+	–	o
Role play	+	o	o	–
Stakeholder observation	o	–	+	o
Work product analysis	o	–	+	–
User feedback analysis	+	–	–	o
Market studies	–	–	o	+
Benchmarking	o	+	–	+

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 quality requirements 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.

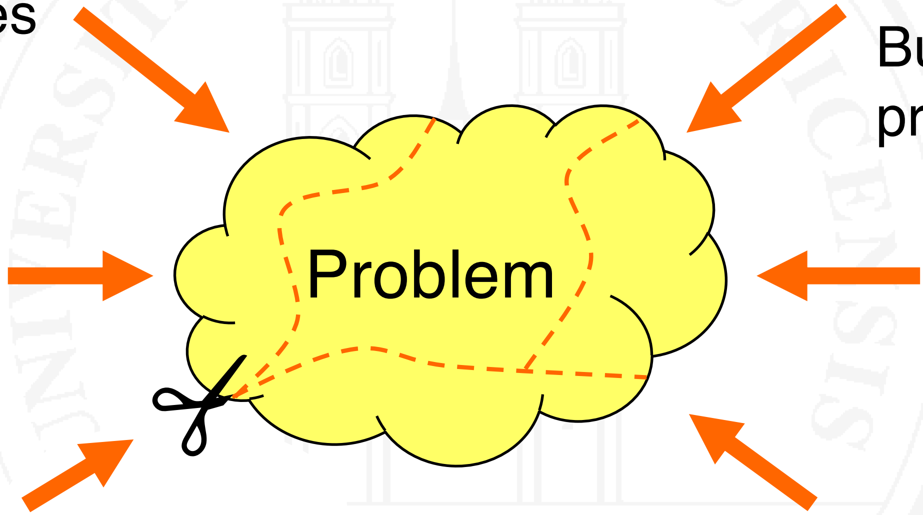
Analysis of elicited information

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 necessarily the best

- Ambiguous
- Imprecise
- Error-prone
- Verification primarily 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]
[Goetz&Rupp 2003]

- 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

Phrase templates

[Mavin et al. 2009]
[Rupp et al. 2009]
[ISO/IEC/IEEE 2018]

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”