Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Raffaela Mirandola

Topic 13
Requirements Engineering

SASIS – SELF-ADAPTIVE SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

sasis.kastel.kit.edu

Course Dates

- Classes

Tentative Dates		
Thu, 10/07/2025	Lecture	Requirements Engineering
Fri, 11/07/2025	Exercise	Sheet 6: Agile Development
Thu, 17/07/2025	**Guest**	**Guest Lecture**
Fri, 18/07/2025	Lecture	Use Cases
Thu, 24/07/2025	Lecture	NLP
Fri, 25/07/2025	Lecture	Software Quality – Reliability
Thu, 31/07/2025	Exercise	Sheet 7: Requirements Engineering and Use Cases
Fri, 01/08/2025	Lecture	Ethic + Wrap-Up

Overview on Today's Lecture

Content: Requirements Engineering

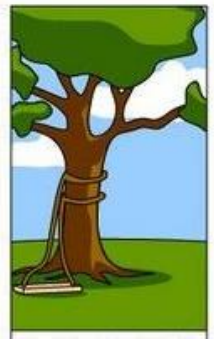
- Repetition & Foundations
- Requirements Elicitation
- Faceted Classification of Requirements
- Requirements Validation & Prioritization
- Tracing Requirements
- Cost and Benefit Considerations

Learning Goals – Be able to

- describe the terms and activities of *Requirements Engineering*
- explain the *requirements problem* according to Jackson
- choose appropriate *elicitation techniques*
- classify and assess requirements
- apply fundamental *guidelines* on specifying natural language requirements
- Give an overview on selected *validation* and *prioritization* techniques

Some Motivation

- ~48% of reasons mentioned for failed software projects related to errors or deficiencies of requirements engineering [Pohl]
  - ➔ *Requirements are missing*
  - ➔ *Requirements are wrong or misunderstood*
  - ➔ *Stakeholders are not involved*



- Solving these problems during later phases is very expensive
  - Boehm estimates a **factor of ten per development phase** [Boehm 2001]

- *You might want to listen to the* Software Engineering Radio — The Podcast for Professional Software Developers

  http://www.se-radio.net/2008/10/episode-114-christof-ebert-on-requirements-engineering

  - ➔ *Christof Ebert's 3 biggest risks in requirements engineering*
    - wrong requirements
    - missing requirements
    - changing requirements

What is a requirement?

- Something wanted or needed

Requirement:
(1) A condition or capability needed by a user to solve a problem or achieve an objective. } needed

(2) 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 documents. } wanted

(3) A documented representation of a condition or capability as in (1) or (2).

[IEEE 610.12-1990]

Software Requirements

[Glinz]

- Ideally, software requirements are described in a way that they are –
  1. _____
  2. _____
  3. _____
  4. _____
  5. _____
  6. _____
  7. _____

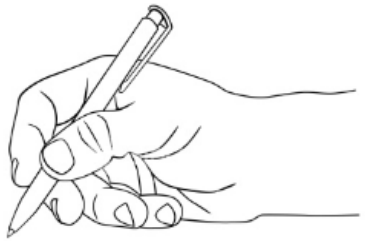
- We distinguish three kinds of requirements, namely –
  1. _____
  2. _____
  3. _____

Software Requirements

- Ideally, software requirements are described in a way that they are –
  1. Adequate (describes what customer wants or needs)
  2. Complete (nothing missing)
  3. Consistent (no contradictions)
  4. Understandable (for developers and stakeholders)
  5. Unambiguous (no wrong interpretation)
  6. Verifiable (can be tested whether system fulfills them)
  7. Suitable for the risk (detail and comprehensiveness based on how likely and how severe to get them wrong)

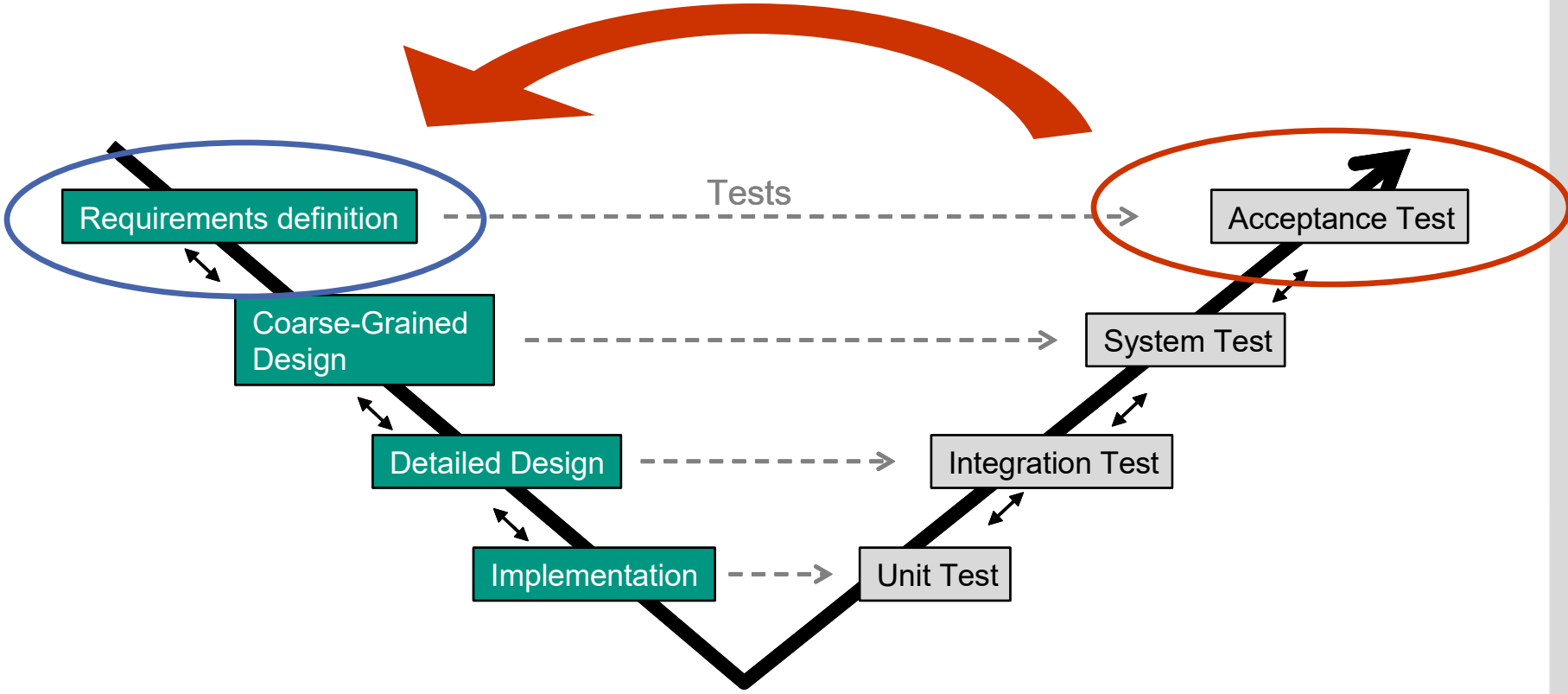
[Glinz]

- We distinguish three kinds of requirements, namely –
  1. Functional requirements
  2. Quality requirements
  3. Constraints



Software Requirements Specification

- The software requirements specification is both starting point and end point for a software development project
  - probably the most important artifact in a project



The Requirements Engineering Process

- Cooperative, iterative and incremental process of –
  - Requirements Elicitation
    - Seeking, capturing, and consolidating requirements from available sources
  - Requirements Documentation
    - Persist the elicited requirements, create Software Requirements Specification (SRS)
  - Requirements Agreement
    - Resolve conflicts
    - Find requirements that are acceptable to all (or most) stakeholders
  - Cross cutting actions:
    - Requirements Validation
    - Requirements Management

[Pohl]

➔ determine and comprehend **all** relevant requirements in **necessary** degree of detail

➔ achieve acknowledgement of requirements by involved stakeholders

Stakeholder

- A person or organisation that (in)directly influences the requirements of a system
  - Users of the system
    - labour union?
  - Operator of the system
  - Purchaser / Sponsor / Controller
  - Software Developer
  - Software Architects
  - Tester
- Identification of relevant Stakeholders and their relationships is critical for RE success
  - missing stakeholders may lead to missing requirements
- Also take care of potential political interests of your stakeholders!

The Requirements Engineer

- Central role in the development process
  - translates between users and developers
  - sometimes also called business analyst
- Needs methodological skills [IREB 2015]
  - thinks analytically
  - has empathy
  - has communication skills
  - is conflict solving
  - has discussion moderation skills
  - is self-confident
  - is convincing
- Is responsible for elicitation, documentation, and agreement
  - maintains the requirements document as well

Viewpoints

The same building.
Different views.

Different viewpoints by different stakeholders must be taken into account.

[Nuseibeh, Kramer und Finkelstein 2003]

Slide taken from Glinz

Consensus and variability

- The viewpoints and needs of different stakeholders may conflict
- Requirements Engineering implies
  - Discovering conflicts and inconsistencies
  - Negotiating
  - Moderating
  - Consensus finding
- But: also determine where variability is needed

Slide taken from Glinz

Systems, machines and context

- Requirements never come in isolation.
  - Requirements specify a system
  - The system may be part of another system
  - The system is embedded in a domain context

Slide taken from Glinz

Systems, machines and context

- Example: Ski lift access control
  - *For every turnstile, the system shall count the number of skiers passing through this turnstile.*
    → Turnstile control software
  - *The system shall provide effective access control to the resort's chairlifts.*
    → Everything: equipment, computers, cards, software
  - *The system shall operate in a temperature range of -30° C to +30° C.*
    → The computer hardware and the devices
  - *The operator shall be able to run the system in three modes: normal (turnstile unlocked for one turn when a valid card is sensed), locked (all turnstiles locked), and open (all turnstiles unlocked).*
    → The access control software for a chairlift

Slide taken from Glinz

Systems, machines and context

- Requirements never come in isolation.
  - Requirements specify a system
  - The system may be part of another system
  - The system is embedded in a domain context

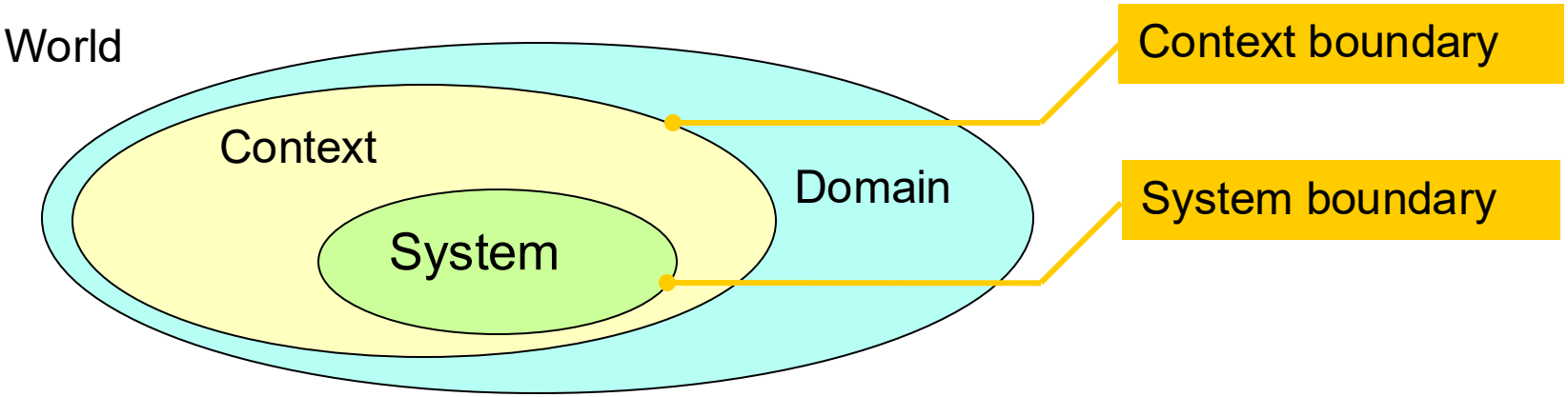
- Requirements need to be framed in a context
- Dealing with multi-level requirements is unavoidable

Slide taken from Glinz

Systems, machines and context

Definition: Context

- 1. In general: The network of thoughts and meanings needed for understanding phenomena or utterances.
- 2. Especially in RE: The part of a system's environment being relevant for understanding the system and its requirements.



Slide taken from Glinz

Systems, machines and context

Definition: System boundary

- The boundary between a system and its surrounding context.

Definition: Context boundary

- Boundary between the context of a system and those parts of the application domain that are irrelevant to the system and its requirements.

- The system boundary separates the system to be developed from its environment
- RE needs to determine the system boundary
- Information outside of the context boundary is not considered

Slide taken from Glinz

Mapping world phenomena to machine phenomena: a major RE problem

- R_{World}: a requirement in the world:

  For every turnstile, the system shall count the number of persons passing through this turnstile.

- R_{System}: Mapped to a requirement for the system to be built:

  The turnstile control software shall count the number of ‘unlock for a single turn’ commands it issues to the controlled turnstile.

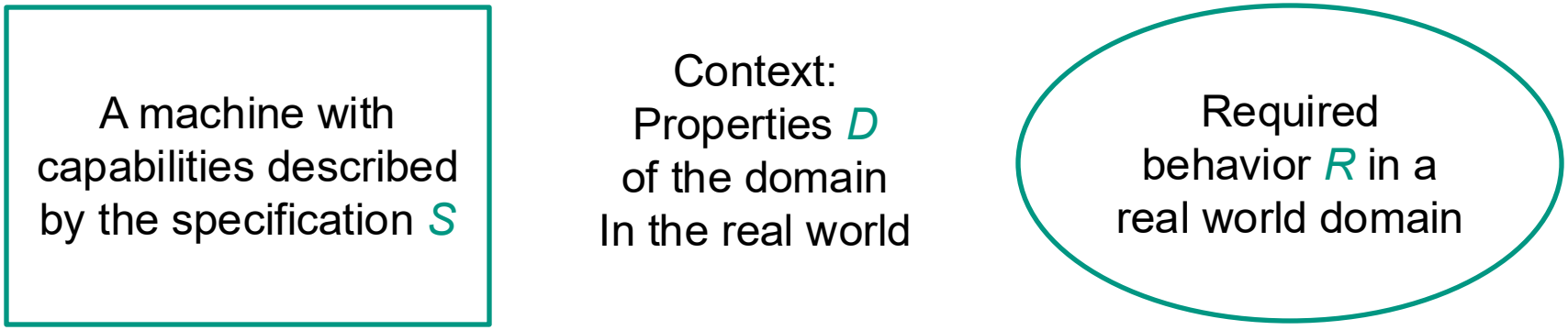
- R_{System} satisfies R_{World} only if these domain assumptions hold:
  - An unlock command actually unlocks the turnstile device
  - When a turnstile is unlocked, a single person passes through it
  - Nobody passes through a locked turnstile (e.g. by crouching down)

Slide taken from Glinz

The world and the machine

[Zave and Jackson 1997]
[Jackson 2005]

- Requirements must hold in the world.
- But we need them to build machines (aka systems).

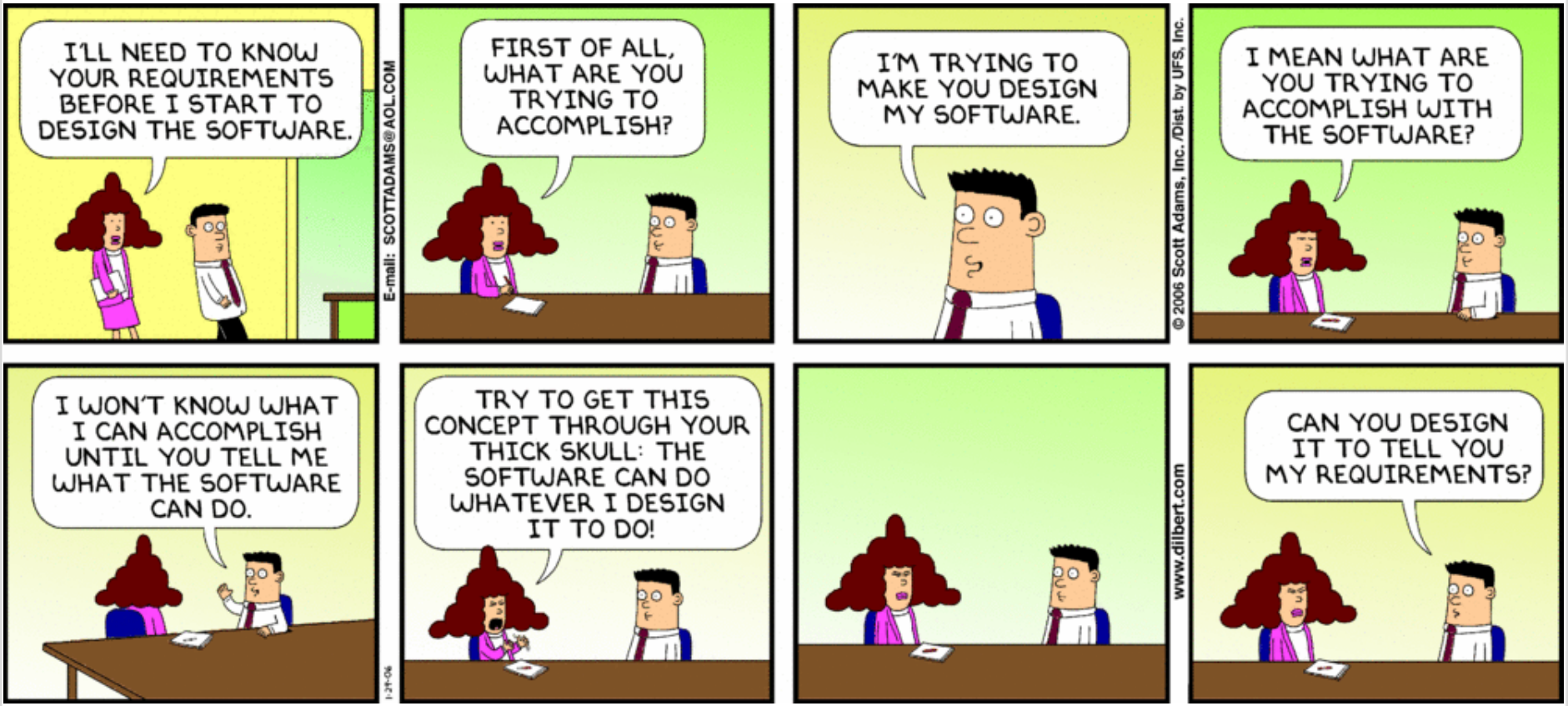


The requirements problem (according to Jackson):

- Given a machine satisfying the specification S and assuming that the domain properties D hold, the requirements R in the world must be satisfied: $S \wedge D \vdash R$

Slide taken from Glinz

Requirements Elicitation



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

Slide taken from Glinz

Requirements Elicitation Techniques

- Which techniques for finding requirements do you know?

Requirements Elicitation Techniques

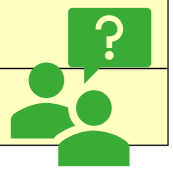
- Ask
  - Interview stakeholders
  - Use questionnaires and polls
- Collaborate
  - Hold requirements workshops
- Build and play
  - Build, explore and discuss prototypes and mock-ups
  - Perform role playing
- Observe
  - Observe stakeholders in their work context
- Analyze
  - Analyze work artifacts
  - Analyze problem/bug reports
  - Conduct market studies
  - Perform benchmarking

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

Slide taken from Glinz

Which technique for what?

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



Slide taken from Glinz

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	+	o	o	–
Prototypes and mock-ups	o	+	–	o
Role play	+	o	o	–
Stakeholder observation	o	–	+	o
Artifact analysis	o	–	+	–
Problem/bug report analysis	+	–	–	o
Market studies	–	–	o	+
Benchmarking	o	+	–	+

Slide taken from Glinz

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:
    “The system must adhere to the GDPR when dealing with person-related data.”
  - Actual requirement:
    “Storing Personal data is only enabled after user indicated consensus for the processing of his / her data. Otherwise, the store-button is displayed as disabled.”

Slide taken from Glinz

Problems Functional vs. Non-functional

- **Non-functional requirements**
  Term often unclear, usage often inconsistent

- Common, but wrong interpretations
  - Non-functional = global
  - Non-functional = soft
  - What the system is supposed to do = functional, how the system is supposed to do it: non-functional
  - Main requirements = functional, supplementary requirements = non-functional
  - Operationally represented = functional, qualitative or quantitative = non-functional requirements

→ These problems can be avoided by considering the underlying concern

Interesting Example [Glinz 2005]

Consider this security requirement

> *(1) "Any unauthorized access to the customer data shall be prevented"*

What about this one?

> *(2) "The access control component shall provide an authentication function that authenticates users by user name and password"*

Concern-based Classification

- *"A concern is a matter of interest in a system.*

- *A concern is a functional or behavioral concern if its matter of interest is primarily the expected behaviour of a system or system component in terms of its reaction to given input stimuli and the functions and data required for processing the stimuli and producing the reaction. […]*

- *A concern is a quality concern if its matter of interest is a quality of the kind enumerated in ISO/IEC 9126 [or in another quality model]."*

-- [Glinz 2007]

How to decide?

Application order

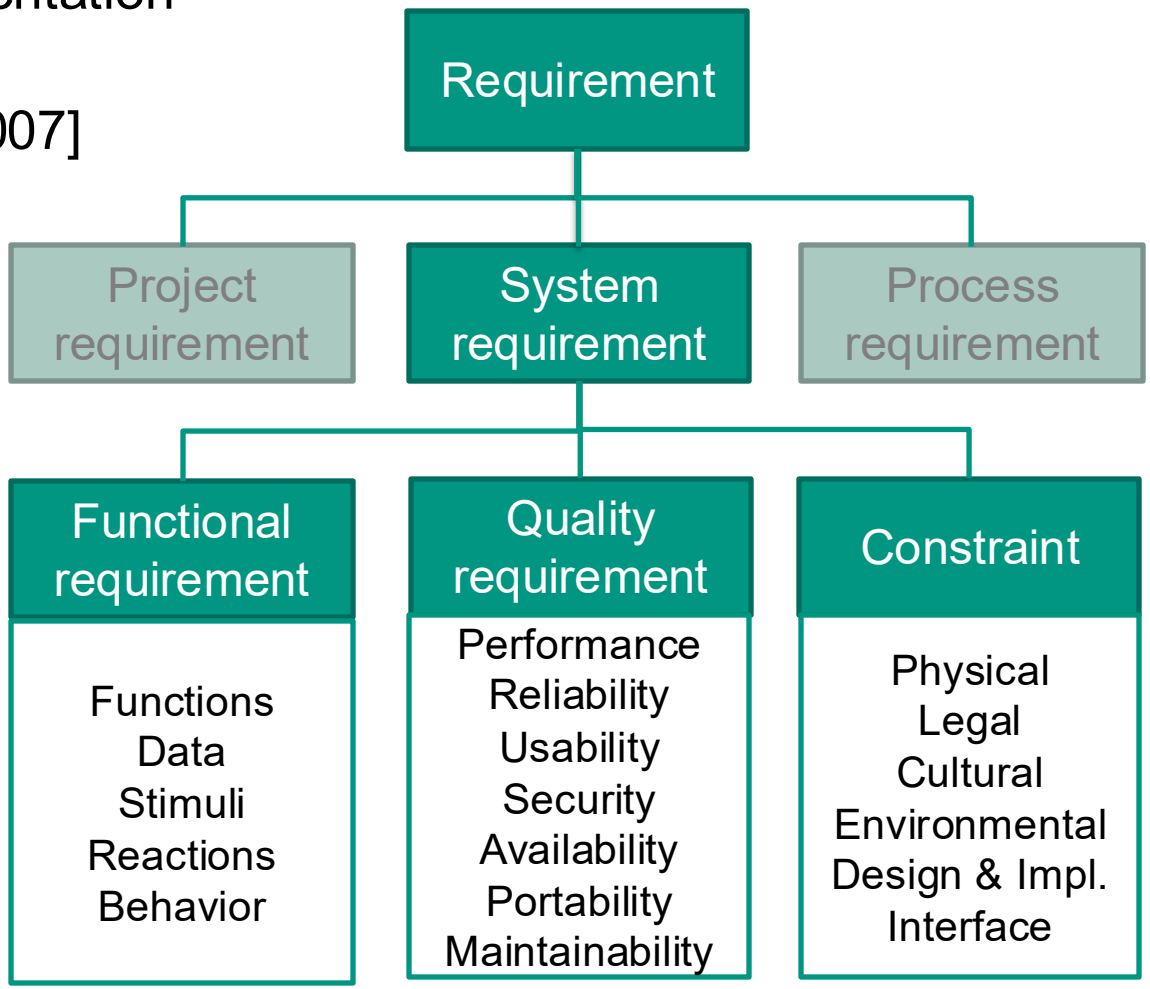
Question	Kind of requirement
What is the underlying concern? Was this requirement stated *primarily* because we need to specify ...	
... some of the system's behaviour, data, input, or reaction to input stimuli – regardless of the way this is done?	functional requirement
... a quality that the system or a component shall have?	quality requirement
... any other restriction about what the system shall do, how it shall do it, or any prescribed solution or solution element?	constraint

Concern-based Classification

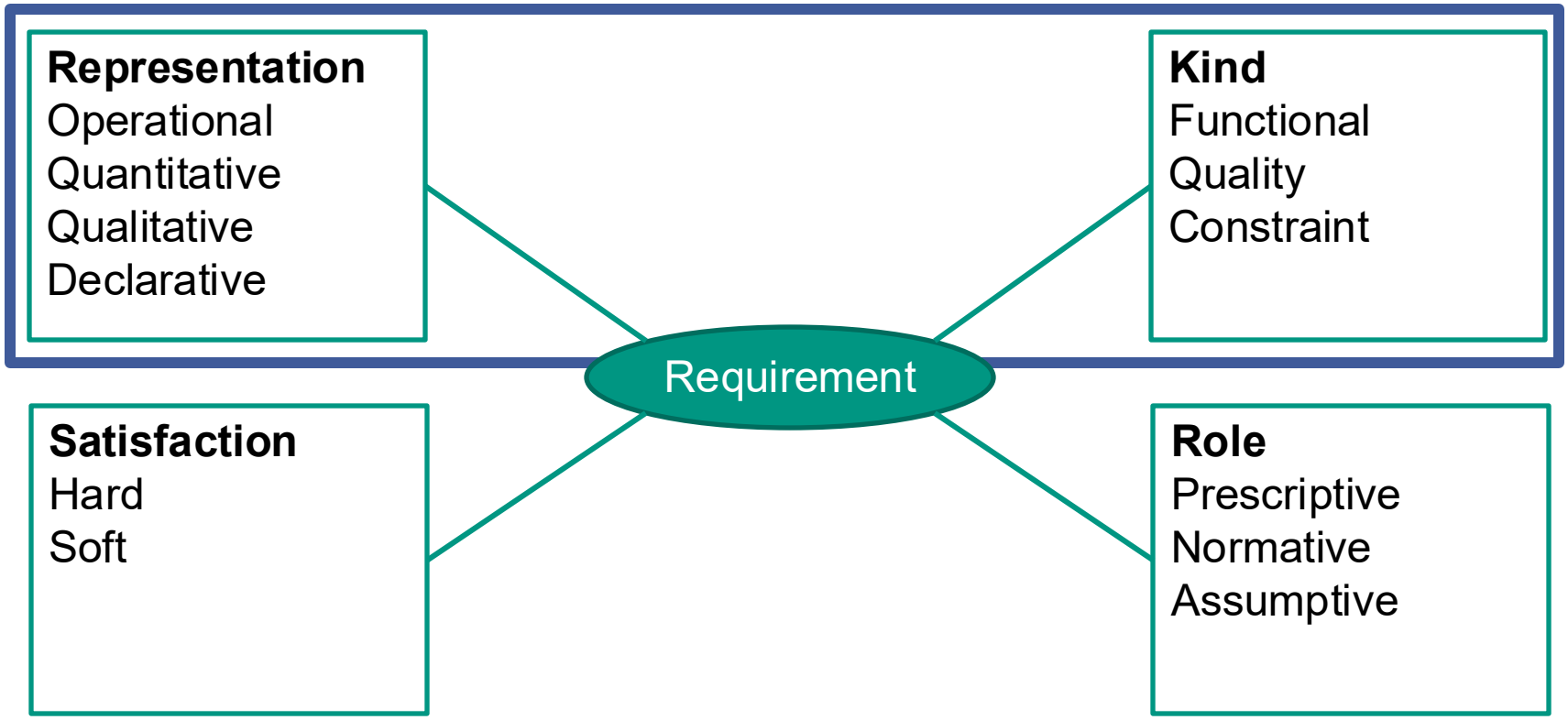
- Independent of representation (see next slide)
- Adapted from [Glinz 2007]

- Functional concern = functional req.
- Quality concern = quality req.

- Constraint: constrains the solution space beyond what is necessary for meeting the given functional and quality requirements.



More Classification Facets



- Kind: based on concern (see previous slides)
- Representation: How is it written down? (next slide)
- Satisfaction: When is it fulfilled?
- Role: describes system-to-be (prescriptive), environment (normative), or actor (assumptive)

Requirements Representation [Glinz 2005]

Representation	Definition	Type of verification
Operational	Specification of operations or data	Review, test, or formal verification
Quantitative	Specification of measurable properties	Measurement (at least on an ordinal scale)
Qualitative	Specification of goals	No direct verification. Either by subjective stakeholder judgement of deployed system, by prototype, or indirectly by goal refinement or derived metrics.
Declarative	Description of a required feature	Review

Interesting Example revisited [Glinz 2005]

Consider this security requirement

- *(1) "Any unauthorized access to the customer data shall be prevented"*

What about this one?

- *(2) "The access control component shall provide an authentication function that authenticates users by user name and password"*

- Functional? No

→ Quality, because security is underlying concern

→ Just represented differently
  - Requirement (1) is represented descriptively
  - Requirement (2) is represented operationally
  - Requirement (2) might be derived from requirement (1)

Examples [Glinz 2005]

Classify the following requirements:

- */R1/ "The system shall compute the sum of all applicable deductions"*
- */R2/ "The system shall be easy to use by casual users"*
- */R3/ "The response time shall be less than 1s on average"*
- */R4/ "The system shall run on PCs featuring at least a 500 MHz CPU and 256 MB main memory."*

- Kind (based on underlying concern):
  Functional, quality, or constraint
- Representation (How is it written down?):
  operational, quantitative, qualitative or declarative
- Satisfaction (When is it fulfilled?):
  hard or soft
- Role (describes system-to-be, environment or actor):
  prescriptive, normative, assumptive

Examples [Glinz 2005]

Classify the following requirements:

- */R1/ "The system shall compute the sum of all applicable deductions"*

- Kind (based on underlying concern):

  _____

- Representation (How is it written down?):

  _____

- Satisfaction (When is it fulfilled?):

  _____

- Role (describes system-to-be, environment or actor):

  _____

Examples [Glinz 2005]

Classify the following requirements:

- */R2/ "The system shall be easy to use by casual users"*

- Kind (based on underlying concern):

  _____

- Representation (How is it written down?):

  _____

- Satisfaction (When is it fulfilled?):

  _____

- Role (describes system-to-be, environment or actor):

  _____

Examples [Glinz 2005]

Classify the following requirements:

- */R3/ “The response time shall be less than 1s on average”*

- Kind (based on underlying concern):

  _____

- Representation (How is it written down?):

  _____

- Satisfaction (When is it fulfilled?):

  _____

- Role (describes system-to-be, environment or actor):

  _____

Examples [Glinz 2005]

Classify the following requirements:

- */R4/ "The system shall run on PCs featuring at least a 500 MHz CPU and 256 MB main memory."*

- Kind (based on underlying concern):

  _____

- Representation (How is it written down?):

  _____

- Satisfaction (When is it fulfilled?):

  _____

- Role (describes system-to-be, environment or actor):

  _____

Examples [Glinz 2005]

	Requirement	Classification
/R1/	The system shall compute the sum of all applicable deductions	
/R2/	The system shall be easy to use by casual users	
/R3/	The response time shall be less than 1 s on average	
/R4/	The system shall run on PCs featuring at least a 500 MHz CPU and 256 MB main memory.	

Examples [Glinz 2005]

	Requirement	Classification
/R1/	The system shall compute the sum of all applicable deductions	Kind: function Representation: operational Satisfaction: hard Role: prescriptive
/R2/	The system shall be easy to use by casual users	Kind: quality -> usability Representation: qualitative Satisfaction: soft Role: prescriptive
/R3/	The response time shall be less than 1 s on average	Kind: quality -> performance Representation: quantitative Satisfaction: soft Role: prescriptive
/R4/	The system shall run on PCs featuring at least a 500 MHz CPU and 256 MB main memory.	Kind: constraint Representation: quantitative Satisfaction: soft or hard Role: prescriptive

More Examples [Glinz 2005]

Requirement	Classification
The applicable deduction are computed according to the formula D = x*y+z which is stated in income tax decree No. 128 of Oct 10th, 2003.	Kind: function or constraint Representation: operational Satisfaction: hard Role: prescriptive or normative
The user must provide accurate data for all input fields of the form.	Kind: quality (of data, user) Representation: declarative Satisfaction: hard Role: assumptive

How to distinguish representations?

- **Operational:**
  A process verb or action verb is used
- **Declarative vs. qualitative:**
  Declarative can be validated by review, qualitative cannot
- **Quantitative:**
  Whether requirement is fulfilled directly measurable with numbers

- Do not confuse *„quality requirement“* with *„qualitative requirement“*

What do I have from this classification?

- Better understanding, less unnecessary discusssions
- Appropriate validation techniques according to representation (→ slide 21)
- The view, that a single requirement can be represented differently, fits to the view on architecture to provide means to realise quality requirements.
- What are just specific operational representations (to be changed) and what are qualiutative / quantitative goals as agreed with other stakeholders

Note:

- Object-oriented design is about refining domain entities and *somehow* adding solution-oriented entities
- Architectural design is about providing means to support quality requirements → changing a requirement to operational representation
  - Hopefully systematically:
    - not just try and error (→ use of patterns)
    - Validatable: through simulation (→ Palladio) or testing
  - Sometimes architectural means are not sufficient (→ algorithmic means may also be necessary to operationalise a requirements)

Classification Summary

- Term “non-functional requirement” often misunderstood
- Often, representation and kind are mixed

- Better:
  - Faceted
  - Separate representation and kind

- But be careful in practice: No common understanding
  - Find out what the understanding is in your project

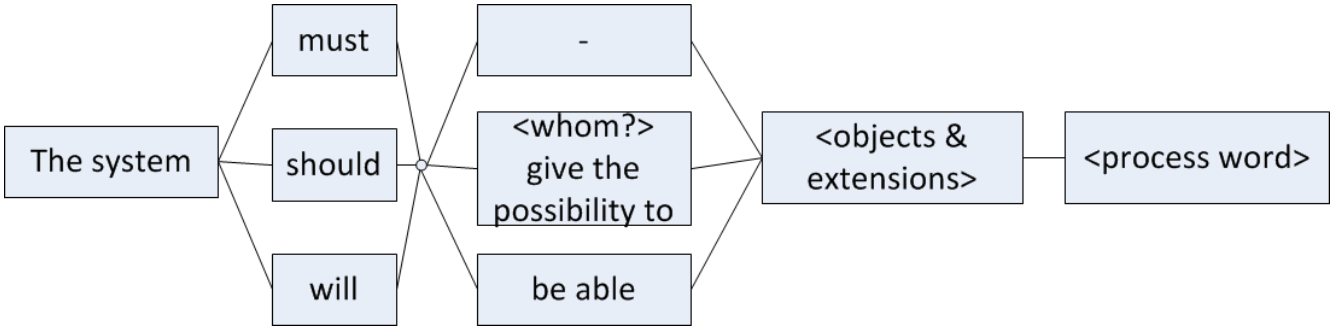
- Learning goal for this course
  - Know kind facet and representation facet
  - Keep in mind that there are other facets (satisfaction and role)

Basic Writing Recommendations

- Most requirements are (initially) captured in natural language
  - as they need to be understandable by users
- Thus, try to follow the following writing guidelines –
  - Short sentences, one clear requirement per sentence
    - *BAD: The Navigation systems navigates to a destination with comfortable usability.*
    - **GOOD: The navigation system must allow in all modes to set the destination.**
  - Use active language that makes clear who is responsible for what
    - *BAD: When the PIN was validated, money can be withdrawn from the account.*
    - **GOOD: When the ATM has validated the PIN, the customer can withdraw money from his account.**
  - Avoid „weak“ words, such as –
    - *effective, consequently, user-friendly, easy, quickly, timely, reliable, appropriate*
  - Maintain a glossary of terms
    - and use them conforming to it

Basic Writing Recommendations II

- Use sentence templates, such as –



- or: [Trigger] Actor Action Object [Condition] [Intel]
  - Trigger: *When the Navigation System is set into on road mode*
  - Actor: *the Navigation System*
  - Action: *displays*
  - Object: *the distance to the destination*
  - Condition: *until another mode is chosen.*

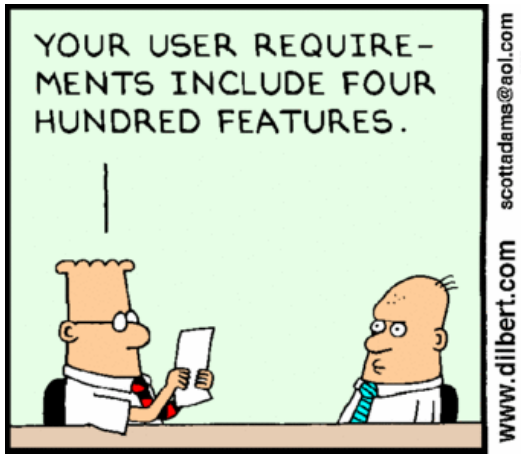
- *or others… see e.g. [Pohl, p. 151], [Ebert 2008, p. 148] if interested*

Traditional Feature Lists

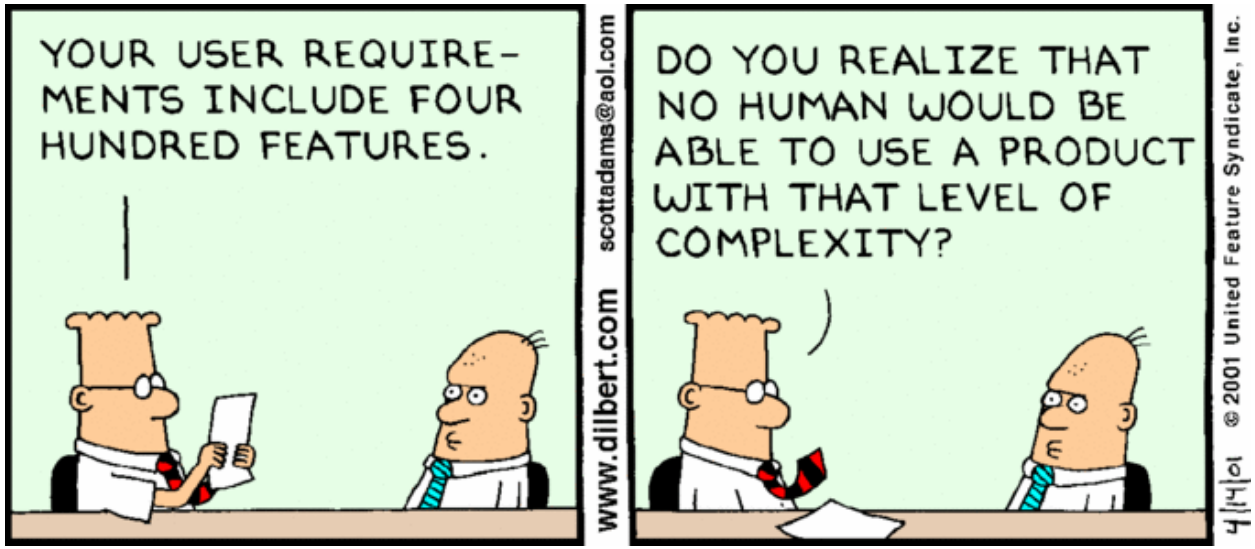
- Traditional requirements analysis methods typically result in detailed, low-level feature lists

ID	Feature
FEAT1.9	The system shall accept entry of item identifiers
….	
FEAT2.4	The system shall log credit payments to the accounts receivable system
….	

- Especially in the context of enterprise applications such long, detailed lists have some drawbacks as they –
  - do not describe requirements in a cohesive way
  - are unstructured and have the feel of a "laundry list"



The Dilbert Solution



Modern Requirements Capture

- Today's most common approaches for writing down requirements are –
  - user stories (agile) & use cases (model-based)
- Both focus on interaction of the user and the software
  - and place the requirements in the context of the stories and goals of using the system
  - chief virtues are simplicity and utility
- Brief history
  - use cases were introduced in 1986 by Ivar Jacobson
  - fleshed out in "Writing Effective Use Cases" by Alistair Cockburn
  - user stories are the predominant approach in agile processes
- Both are not always the most appropriate approach
  - some applications still call for a feature-driven approach
    - e.g. application servers, data products, middleware or back end systems
- Both are easy to link to other models (sequence diagrams) through naming conventions and through them to code entities.
- **→ Traceability nearly for free!**

→ More in next lecture on Use Cases

Requirements Validation

- *Remember: Requirements errors are expensive*
  - find errors in requirements as early as possible

- Validation (in the context of RE): *Am I building the right system?*
  - *…to solve my stakeholders' problems? Are the requirements right?*
- Verification: *Am I building the system right?*
  - correctness (relation between two documents / artefacts)
  - *Note that a Review on Requirements is also a verification technique (but concerned with "Am I building the right system").*

- Validation of requirements artefacts (output)
  - Ambiguities, incompleteness, inconsistencies
- Validate context aspects (input)
  - Wrong or missing context information
- Validate RE process
  - Missing steps, stakeholders

➔ *Without a formal req. model, only reviews and prototypes help!*
  - *Simulation and verification when formal model is available*

Requirements Validation Techniques

- Review
  - Main means for requirements validation
  - Walkthrough: author guides experts through the specification
  - Inspection: Experts check the specification
  - Author-reviewer-cycle: Requirements engineer continuously feeds back requirements to stakeholder(s) for review and receives feedback
- Requirements Engineering tools
  - Help find gaps and contradictions
- Acceptance test cases
  - Help disambiguate / clarify requirements

Slide taken from Glinz

Requirements Validation Techniques

- Simulation/Animation
  - Means for investigating dynamic system behavior
  - Simulator executes specification and may visualize it by animated models
- Prototyping
  - Lets stakeholders judge the practical usefulness of the specified system in its real application context
  - Prototype constitutes a sample model for the system-to-be
  - Most powerful, but also most expensive means of requirements validation
- Formal Verification / Model Checking
  - Formal proof of critical properties

Slide taken from Glinz

Requirements Priorization

- Requirements may be prioritized with respect to various criteria, for example
  - Necessity
  - Cost of implementation
  - Time to implement
  - Risk
  - Volatility
- Prioritization is done by the stakeholders
- Only a subset of all requirements may be prioritized
- Requirements to be prioritized should be on the same level of abstraction

Slide taken from Glinz

Selected Requirements Prioritization

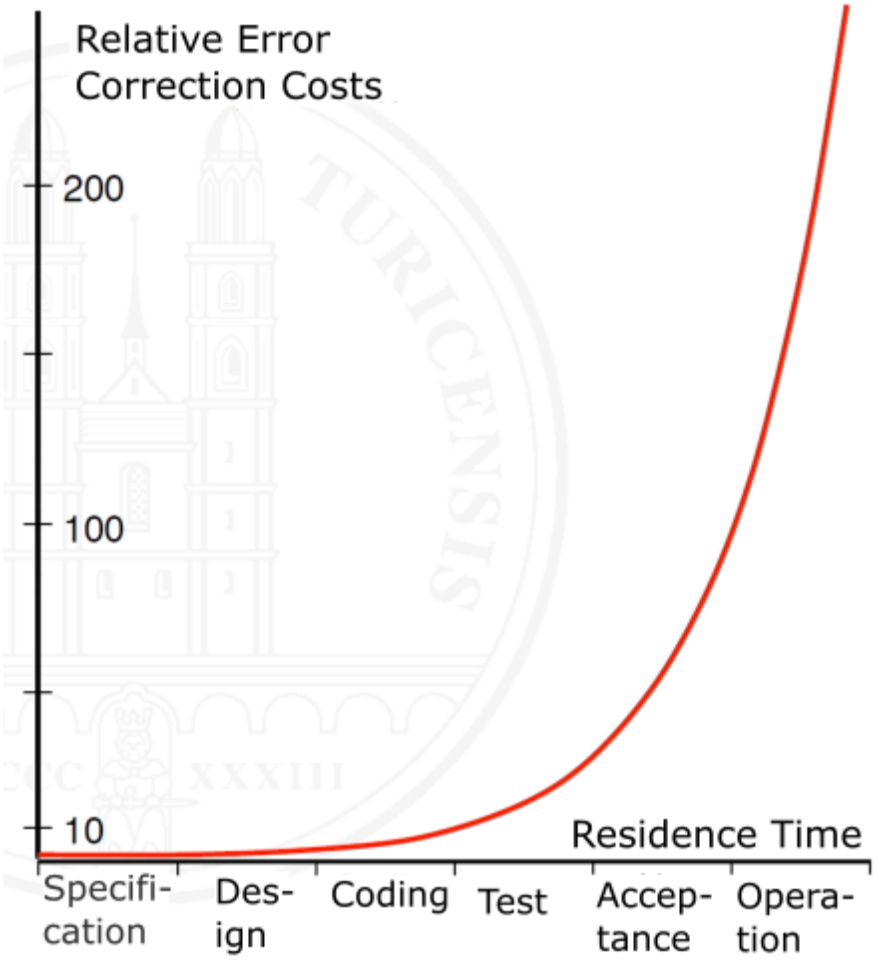
Single criterion prioritization

- Simple ranking
  - Stakeholders rank a set of requirements according to a given criterion
- Assigning points
  - Stakeholders receive a total of n points that they distribute among m requirements
- Prioritization by multiple stakeholders may be consolidated using weighted averages. The weight of a stakeholder depends on his/her importance

Slide taken from Glinz

RE Benefit

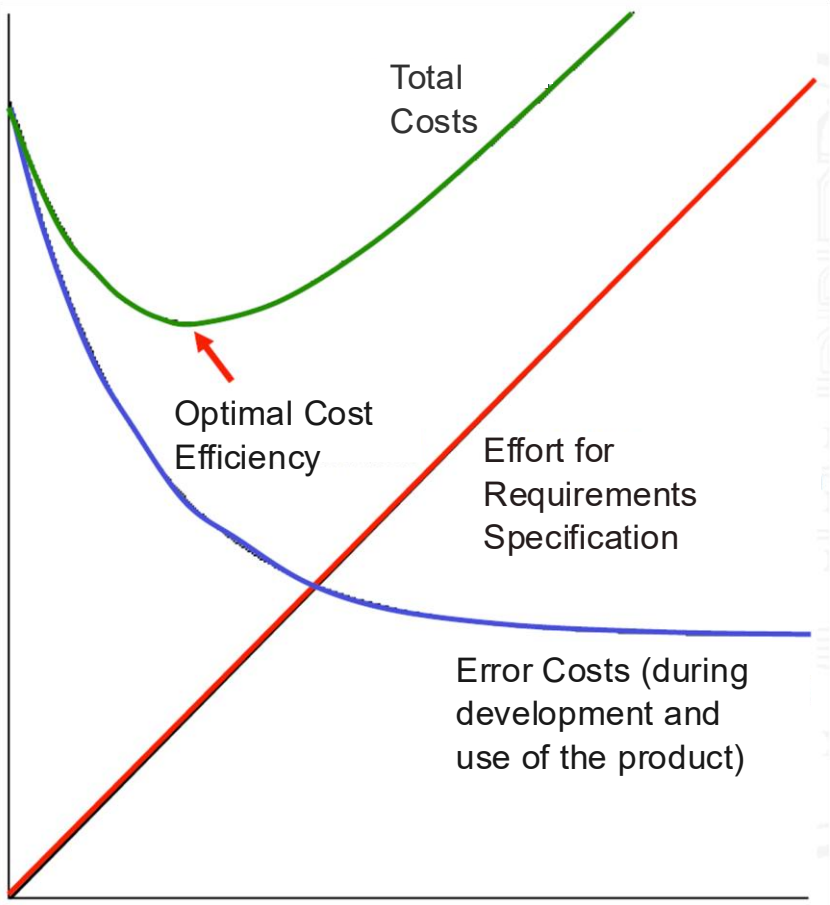
- Related to Boehm's first law
- Cost for removing errors depends on how long it stays in the software
- Good RE avoids requirements errors and thus saves cost for removing errors later



From [Glinz]

RE Cost

- Cost to remove errors vs cost to specify requirements



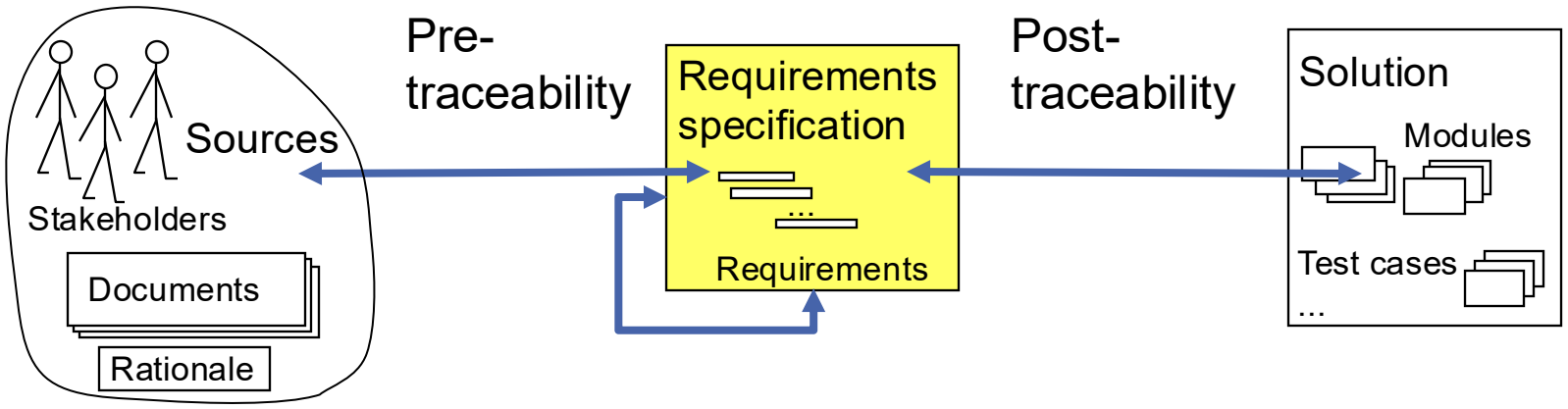
Also not useful to specify requirements down to the last detail

→ trade-off between cost to specify and expected gain (in terms of avoided errors)

From [Glinz]

Traceability

- Definition: Traceability
  - = The ability to trace a requirement
    - (1) back to its origins,
    - (2) forward to its implementation in design and code,
    - (3) to requirements it depends on (and vice-versa).
    - Origins may be stakeholders, documents, rationale, etc.



Slide taken from Glinz

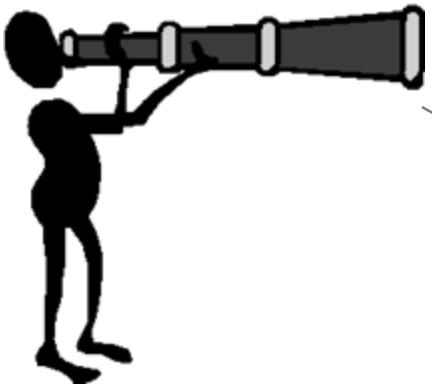
Establishing and maintaining traces

- Manually
  - Requirements engineers explicitly create traces when creating artifacts to be traced
  - Tool support required for maintaining and exploring traces
  - Every requirements change requires updating the traces
  - High manual effort; cost and benefit need to be balanced
  - Use cood naming conventions consistently for cross-artefact traceability (again use cases to sequence diagrams, as this is a 1:1 relationship)
- Automatic
  - Automatically create candidate trace links between two artifacts (for example, a requirements specification and a set of acceptance test cases)
  - Uses information retrieval technology
  - Requires manual post processing of candidate links

Slide taken from Glinz

Conclusion

- Various techniques for capturing requirements
- Be aware of different classes of requirements
- Writing recommendations
- Validating requirements is crucial
- Prioritization of requirements is necessary
- Tracing requirements can be helpful
- Cost and benefit of RE



Use Case Writing

References

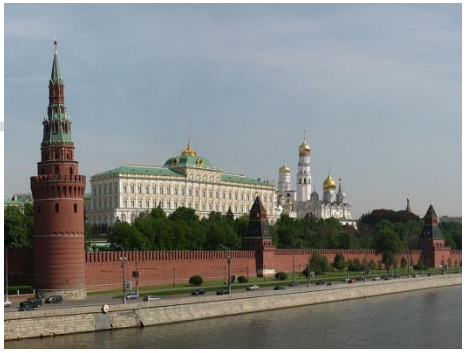
- [Ambler] S. Ambler, *The Object Primer: Agile Model-Driven Development with UML 2.0,* Cambridge University Press, 2004
- [Boehm 2001] Barry Boehm and Victor Basili, Software Defect Reduction Top 10 List, Computer, January 2001, pp. 135-137
- [IREB 2015] Syllabus IREB Certified Professional for Requirements Engineering - Foundation Level - Version 2.2, March 1st 2015
- [Glinz] Vorlesungsfolien “Requirements Engineering I”, Institut für Informatik, Universität Zürich (abgerufen für Herbstsemester 2011)
- [Glinz 2005] Martin Glinz, Rethinking the Notion of Non-Functional Requirements, Proceedings of the Third World Congress for Software Quality, Munich, Germany, September 2005.
- [Glinz 2007] Martin Glinz, On Non-functional Requirements, Proceedings of the 15th IEEE International Requirements Engineering Conference, Delhi, India, 2007
- [Patton] Patton, J. (2005) How You Slice it, Better Software.
- [Pohl] Pohl, K. (2007), Requirements Engineering: Grundlagen, Prinzipien, Techniken, dpunkt, Heidelberg
- [Lamsweerde] Lamsweerde, A. V. (2001), Goal-oriented requirements engineering: A guided tour
- [Larman] C. Larman, *Applying UML and Patterns (3rd ed.),* Prentice Hall, 2004
- [Kano] Kano, N.; Seraku, N.; Takahashi, F. & Ichi TSUJI, S. (1984), 'Attractive Quality and Must-Be Quality',

References

- M. Jackson (2005). Problem Frames and Software Engineering. Information and Software Technology 47(14). 903–912.
- P. Zave, M. Jackson (1997). Four Dark Corners of RequirementsEngineering. ACM Transactions on Software Methodology 6(1). 1–30.
- J. Goguen, C. Linde (1993). Techniques for Requirements Elicitation. Proceedings of the First IEEE International Symposium on Requirements Engineering (RE'93) San Diego, California, USA. 152–164.
- E. Gottesdiener (2002). Requirements by Collaboration: Workshops for Defining Needs. Boston: Addison-Wesley.
- A.M. Hickey, A.M. Davis (2003). Elicitation Technique Selection: How Do Experts Do It? Proceedings 11th IEEE International Requirements Engineering Conference (RE'03), Monterey Bay, USA. 169–178.
- O. Dieste, N. Juristo, F. Shull (2008). Understanding the Customer: What Do We Know about Requirements Elicitation? *IEEE Software* **25**(2). 11–13.
- D. Zowghi, C. Coulin (2005). Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In A. Aurum, C. Wohlin (eds.) *Engineering and Managing Software Requirements*. Berlin: Springer. 19–46.
- T.L. Saaty (1980). The Analytic Hierarchy Process. New York: McGraw Hill.
- K.E. Wiegers (1999). Software Requirements. Redmond, WA: Microsoft Press.

APPENDIX

Priority: MoSCoW



[Минеева Ю., Wikipedia]

- … is the capital of Russia ☺
- … but also a way of prioritizing software requirements
  1. MUST have
     - a critical requirement that must be present in the product
  2. SHOULD have
     - important, but not absolutely necessary requirement
  3. COULD have
     - nice to have requirement that could increase customer satisfaction
  4. WON‘T have / WOULD like
     - requirement of relatively low importance that still increases business value

➔ [IEEE-830]: *Mandatory, Optional, Nice to have…*

Priority: Cost Value Analysis

- One dimensional prioritization is often not sufficient
  - contrastive pairs are often more helpful
  - using value/importance vs. –
    - cost / time
    - risk
    - volatility
    - etc…
- for example

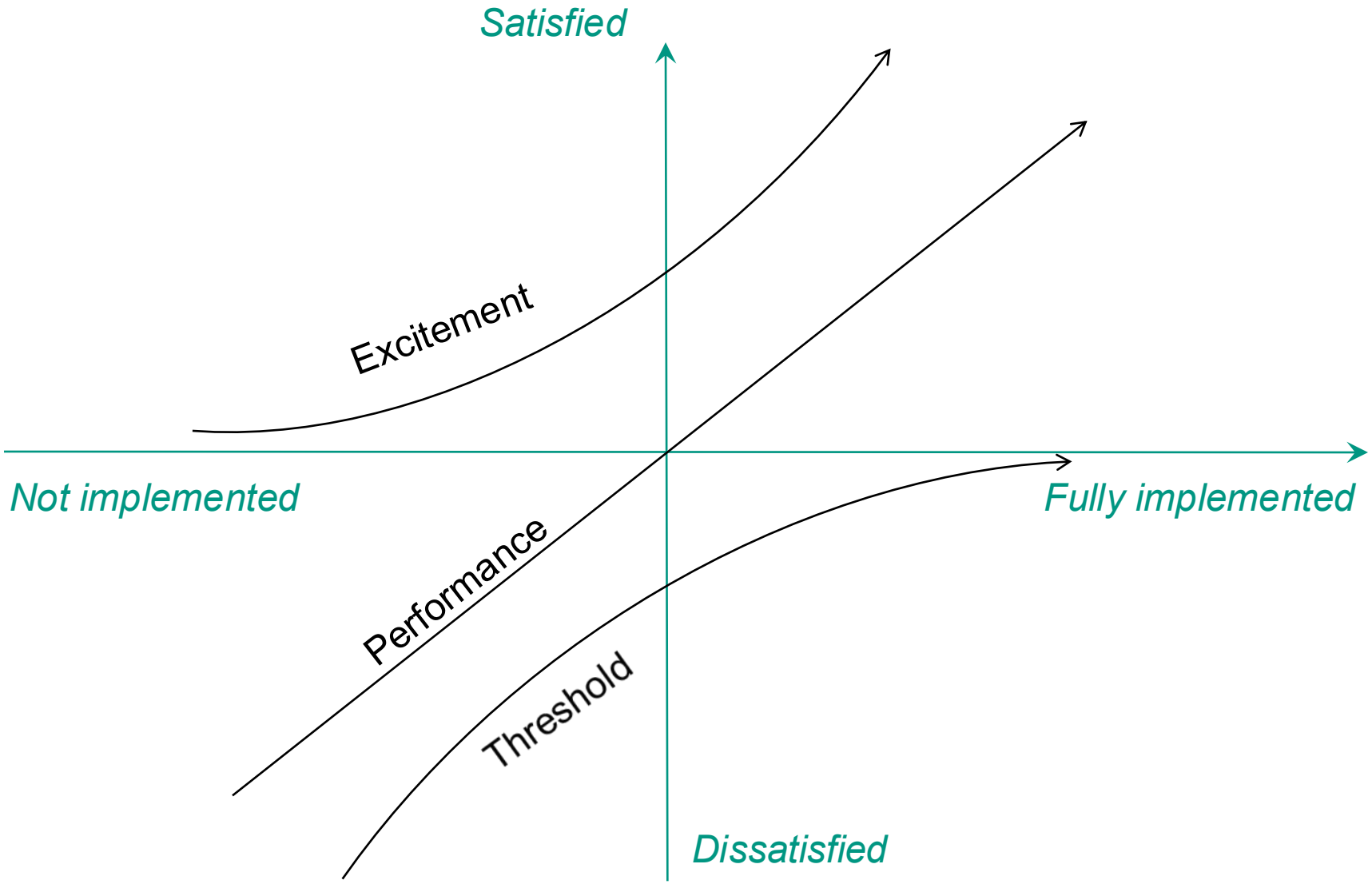
Reqmnt.	Value	Cost	-> Prio.
A	4	2	2
B	3	3	1
C	3	2	1.5

➔ *approach is context-dependent*

Priority: The Kano Model [Kano84]

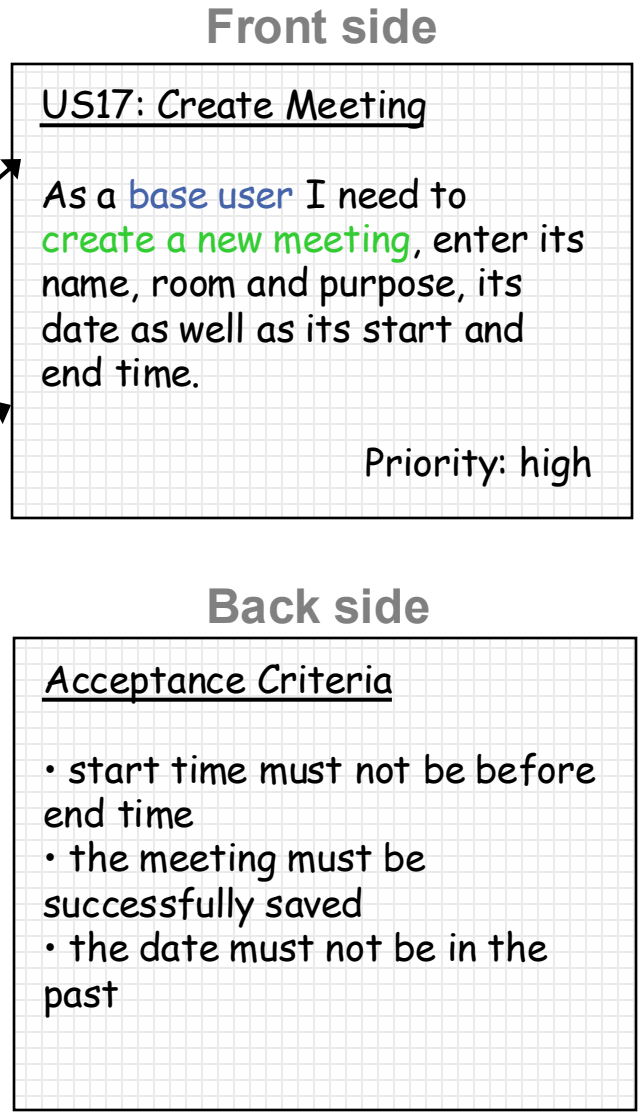
- Categorisation for requirements
  - Attractive Quality *(Excitement)*
    - features that make a system attractive to use
    - but are not expected by the user
  - One-dimensional Quality *(Performance)*
    - critical key functionality
    - explicit requirements that are often advertised
  - Must-be Quality *(Threshold)*
    - basic attributes taken for granted
    - often implicit requirements that are not spoken out
  - Indifferent Quality
    - features that do not influence customer satisfaction
  - Reverse Quality
    - people have different tastes

The Kano Model



… with User Stories [Cohn04]

- User Stories are collected on index cards
  - not suitable for UI requirements
- They describe requirements from the **end-user's point of view**
  - they comprise
    - a **name** (and ideally an ID)
    - a brief **textual description**
    - **acceptance criteria**
- The textual description should contain –
  - the user's role
  - the goal of the story
  - optionally the benefit of the requirement



Agile Requirements Elicitation

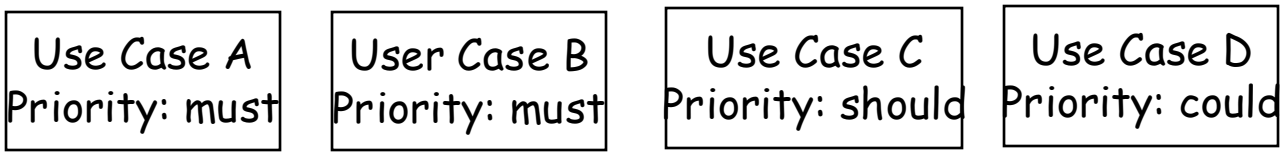
1. The Product Owner initially populates the product backlog with **coarsely described requirements**
   - derived from the product concept and discussed with the customer
   - the goal is to quickly collect some requirements
2. The Product Owner **clusters** the requirements according to their overall themes
3. Product Owner and team **prioritize** the requirements
   - usually according to business value and perhaps risk
4. The Product Owner **refines** the requirements with the highest priorities
   - in requirements workshops with the customer
     - and the team as far as possible
   - the product backlog should now contain enough information to start with the first sprint
5. The requirements are updated, refined or even removed as needed

Release Planning Again

- Does building the highest priority features first really deliver the best system increments?
  - or in other words: *how to assign requirements to meaningful releases?*

➔ A priority-driven assignment of requirements often leads to unusable intermediate releases

- since low-priority requirements are often needed to “hold the software together”



➔ A one-dimensional prioritized list may not be sufficient to organize releases

➔ *“Design your project in working layers [of requirements] to avoid half-baked incremental releases”* [Patton05]

Story Maps [Patton05]

- The idea of story mapping is to arrange features (user stories) in two dimensions
  - according to their –
    - priority / criticality
    - natural usage sequence

Story Mapping Example

1. Start by collecting user stories as usual
   - for example for a software for small retailers
     1. Create purchase order for vendor
     2. Receive shipment from vendor
     3. Print price tags for received items
     4. Sell items
     5. Return and refund items
     6. Analyze sales
2. Detail and prioritize these user stories

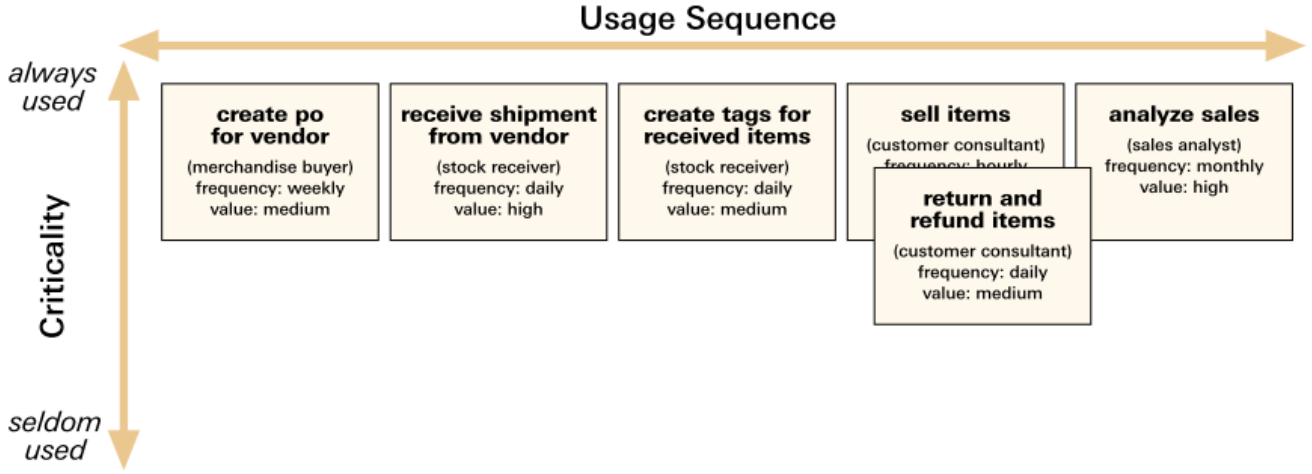
> User Story 1: As a merchandise buyer I would like to create a purchase order (po) for a vendor.
>
> frequency: weekly
> value: medium

-> in the context of a shop management system

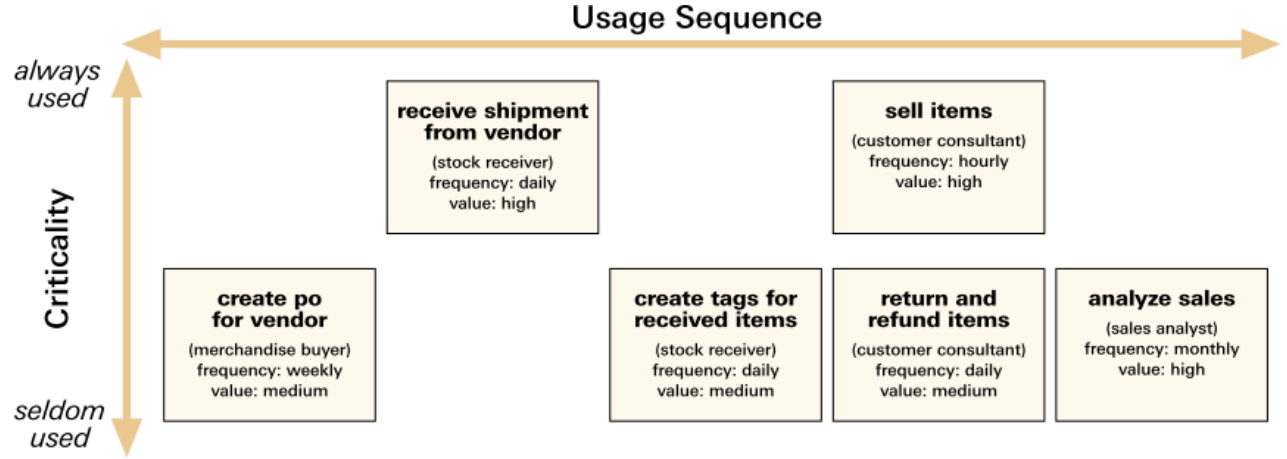
3. Arrange them in their natural sequential order

Creating the Story Map

- User Stories arranged by sequence (step 3)

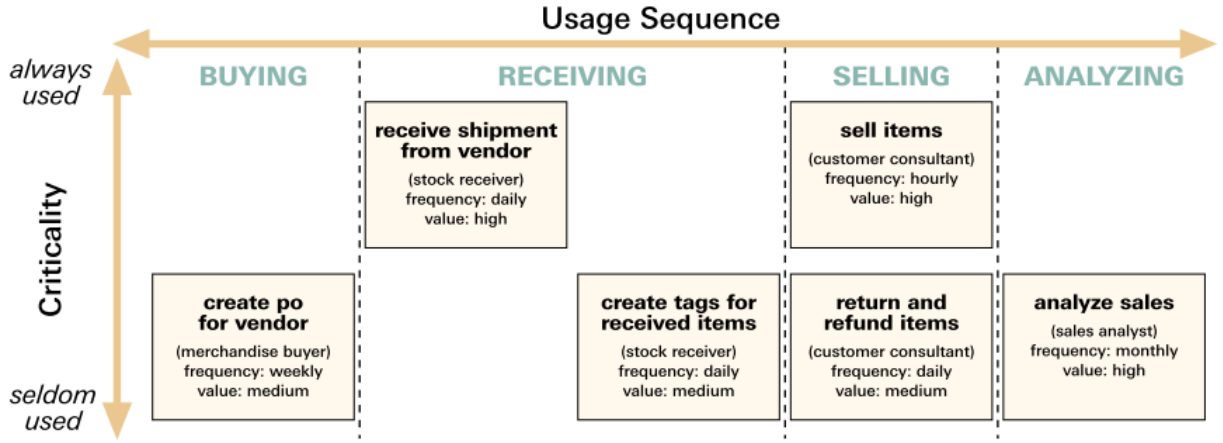


4. Sort them according to criticality afterwards

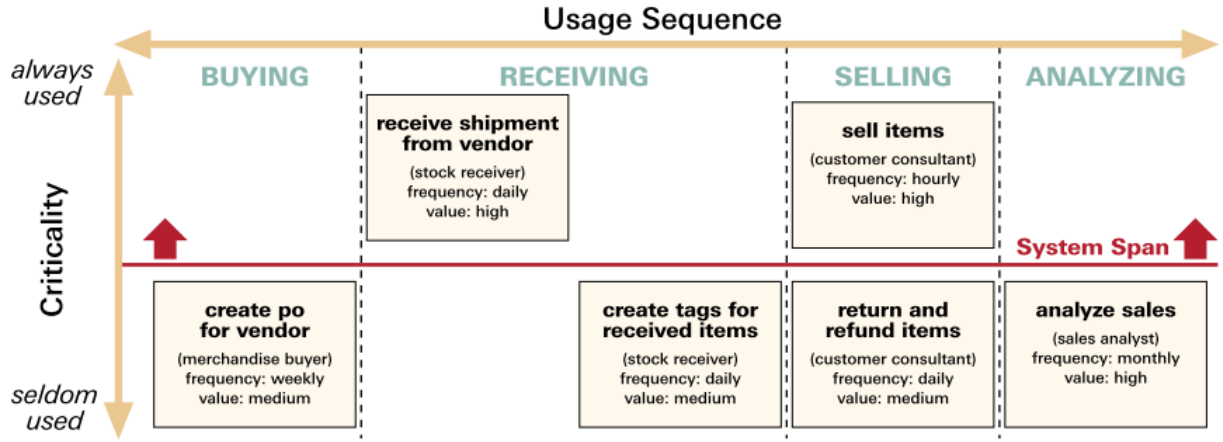


Creating the Story Map II

5. Identify business processes

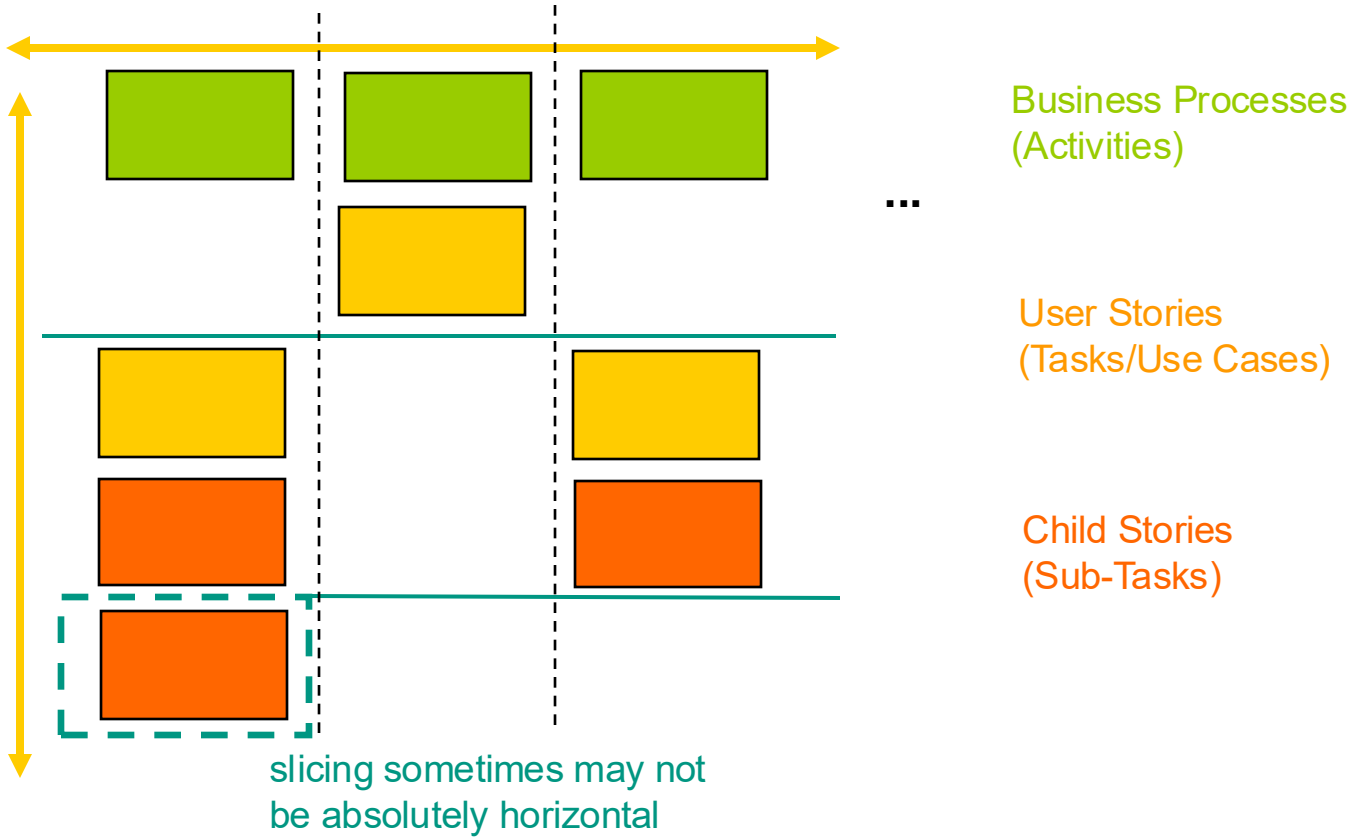


6. Identify meaningful system releases by horizontal slicing



Practical Hints

- You may need to decompose user stories into child stories
  - use differently colored cards for them



➔ this model may be helpful for other I&I approaches as well