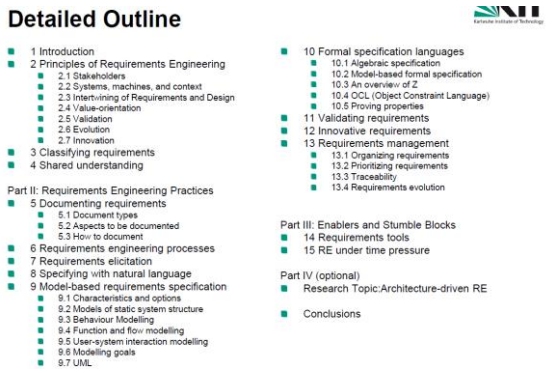
Requirements Engineering

Lukas Bach - lbach@outlook.de - lukasbach.com



Detailed Outline

- 1 Introduction
- 2 Principles of Requirements Engineering
  - 2.1 Stakeholders
  - 2.2 Systems, machines, and context
  - 2.3 Intertwining of Requirements and Design
  - 2.4 Value-orientation
  - 2.5 Validation
  - 2.6 Evolution
  - 2.7 Innovation
- 3 Classifying requirements
- 4 Shared understanding

Part II: Requirements Engineering Practices

- 5 Documenting requirements
  - 5.1 Document types
  - 5.2 Aspects to be documented
  - 5.3 How to document
- 6 Requirements engineering processes
- 7 Requirements elicitation
- 8 Specifying with natural language
- 9 Model-based requirements specification
  - 9.1 Characteristics and options
  - 9.2 Models of static system structure
  - 9.3 Behaviour Modelling
  - 9.4 Function and flow modelling
  - 9.5 User-system interaction modelling
  - 9.6 Modelling goals
  - 9.7 UML
- 10 Formal specification languages
  - 10.1 Algebraic specification
  - 10.2 Model-based formal specification
  - 10.3 An overview of Z
  - 10.4 OCL (Object Constraint Language)
  - 10.5 Proving properties
- 11 Validating requirements
- 12 Innovative requirements
- 13 Requirements management
  - 13.1 Organizing requirements
  - 13.2 Prioritizing requirements
  - 13.3 Traceability
  - 13.4 Requirements evolution

Part III: Enablers and Stumble Blocks

- 14 Requirements tools
- 15 RE under time pressure

Part IV (optional)

- Research Topic:Architecture-driven RE
- Conclusions

1 Fundamentals

1.1 Introduction

1.1.1 Definition Requirement

1. A need perceived by a stakeholder
2. A Capability/property that a system shall have
3. A documented representation of a need, capability or property.

1.1.2 Definition Requirements Spec

Systematically represented collection of requirements, typically for a system or component, that satisfies given criteria.

- Lastenheft: Customer Specs
- Pflichtenheft: System Specs

Risk-based RE: “How much RE do we need such that the risk of deploying the wrong system becomes acceptable?”. The effort spent for Requirements Engineering shall be inversely proportionla to the risk that one is willing to take.

1.2 7 Principles of RE

1.2.1 Stakeholders

Person/Org with direct or indirect influence on the systems requirements.

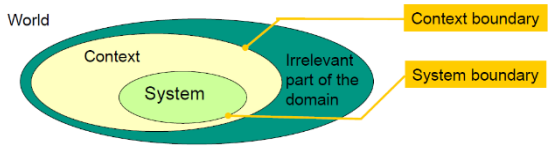
RE also implies conflict handling.

1.2.2 Systems, machines and context

System of systems, find system contexts and deal with multi-level requirements.

- System boundary: Between system and its context
- Context boundary: Between system context and application domain parts that are irrelevant for the system.

RE needs to determine system boundary and ignore information outside the context boundary.



Context models: Determine level of spec (no system internals, only black boxes), model actors that interact with the system, model interaction between system and its actors, model interaction among actors, represent result graphically.

1.2.2.1 The requirements problem (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: $S \wedge D \vdash R$.

[TODO: Mini Exercise at F01.36]

1.2.3 Intertwining of Goals, Requirements and Design

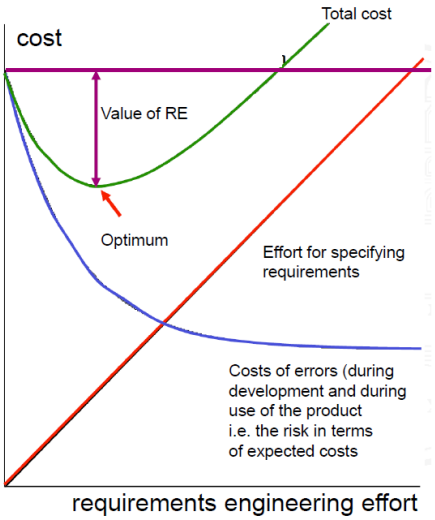
Specs and implementation are inevitably intertwined, full Specs can’t be defined beforehand. Also design decisions affect requirements, which in turn create more design decisions.

Typically, requirements on layers: Business, System, Software.

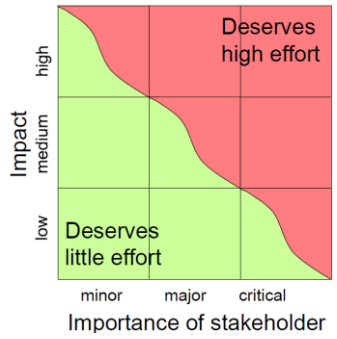
Requirements and Design should be documented seperately, but What vs How does not work. So: Distinguish operationally: If statement owned by stakeholders, it’s a requirement, if it’s owned by supplier, it’s design.

1.2.4 Value-Orientation

Good RE must create value, adapt the effort to the projects risk.

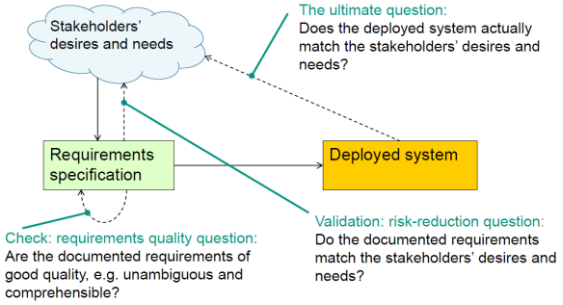


Assessing risk:



Other factors for assessing risk: Distinctivenes, shared understanding, reference systems, length of feedback-cycle, kind of customer-supplier relationship, certification required.

1.2.5 Validation



1.2.6 Evolution

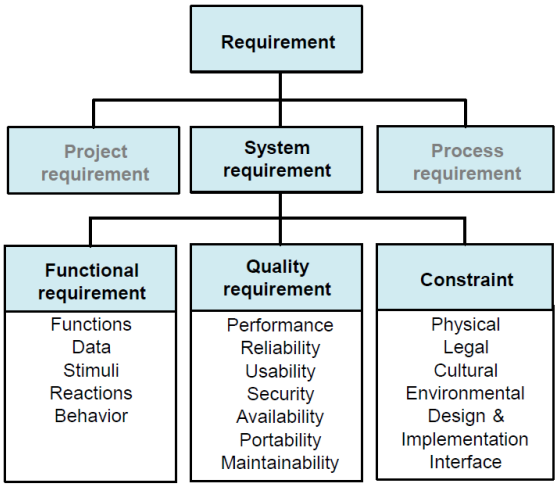
Problem: Keep Reqs stable while permitting reqs to change.

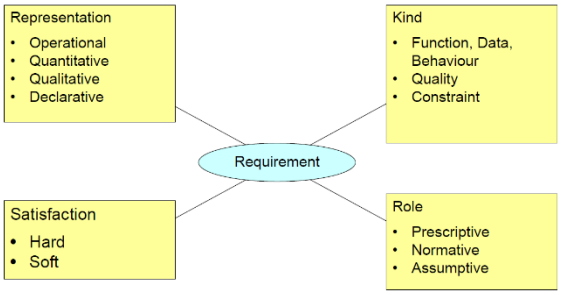
Solution: Short dev cycles, explicit req management.

1.2.7 Innovation

Don't just automate and satisfy customers, make them happy.

1.3 Classifying requirements





Examples see F01.52ff.

Form	Definition	Validation levels
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

How to distinguish representations:

- Operational: A process verb or action verb is used
- Declarative vs Qualitative: Declarative can be validated by review, qualitative cannot
- Qualitiative requirements cannot have hard satisfaction

Classification to role:

- Prescriptive: The sensor value shall be read every 100ms. (classic req for system-to-be)

- Normative: A SSN uniquely identifies a patient. (norm for the system-to-be)
- Assumptive: The operator shall acknowledge every alarm. (Required behaviour of an actor).

Excersize: F01.64.

1.4 Shared understanding

TODO

Achieve successful software dev by:

- Achieving shared understanding by explicit specs as far as needed
- Relying on implicit shared understanding of relevant information as far as possible
- Determining the optimal amount of explicit specs by striking a proper balance between cost/benefit of explicit specs.

TODO Exercise F01.76

2 Requirements Engineering Practices

2.1 Document Requirements

2.1.1 Document Types

- Stakeholder requirements spec (Lastenheft, written by customer domain experts)
- System requirements spec (Pflichtenheft, by supplying req engineer)
- Software requirements spec (if system is pure software)
- Business requirements spec
- Collection of user stories/task descriptions, used for agile processes

2.1.2 Aspects to be documented

See 1.3, types of system requirements.

2.2 Req Engineering processes

2.2.1 Principal Tasks

- Requirements Elicitation (Gewinnung)
- Requirements Documentation
- Requirements Agreement
- During all of those: Validation, Management

2.2.2 Dimensions

- Linear vs evolutionary processes
- Prescriptive vs Explorative vs COTS-driven
  - Prescriptive: Req specs are a binding contract, functionality determines cost and deadlines, development may be outsourced.
  - Explorative: Goals are known, reqs have to be explored. Stakeholders strongly involved, deadlines and cost constrain functionality.
  - COTS-driven: System will be implemented with Commercial-Off-The-Shelf-Software. Often only reqs not covered by COTS are specified.
- Customer-specific vs Martket-oriented

Typical combinations:

- Participatory: evolutionary, exploratory, customer specific
- Contractual: linear, prescriptive, customer-specific
- Product-orientated: evolutionary, mostly explorative, market-oriented
- COTS-aware: evo/linear, COTS-driven, customer-specific

2.3 Requirements elicitation

Find reqs, make stakeholders happy, work value-driven and risk-driven.

Information sources: Stakeholders, context, observation, documents, existing systems.

Stakeholder analysis: Identify and classify stakeholders (critical, major, minor)

Context analysis: Determine system's context and context boundary, and context constraints.

Goal analysis: Main goals, relations between them?

Technique	Suitability for			
	Express needs	Demonstrate opportunities	Analyse 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
Artefact analysis	o	–	+	–
Problem/bug report analysis	+	–	–	o
Market studies	–	–	o	+
Benchmarking	o	+	–	+

TODO Papers

TODO F2.47ff

2.4 Specifying with natural language

2.4.1 Rules

- Use active voice and defined subjects
- Use terms from glossary
- Define precise meanings for auxiliary verbs (shall, should, must, may…)
- Check for nouns with unspecific semantics (the data, the customer…)
- Reference for comparatives: better THAN
- More: F63

Use Phrase templates (and agile stories, F2.65)

Avoid redundancy, except for abstraction purpose, avoid local redundancy (never ever).

2.5 Model-based req spec

(Works only really for functional requirements)

Models: Entity-relationship models, class and object models, component models.

2.5.1 Models of static system structure

- Entity-relationship models
  - Used for data-modelling
  - Easy to model, but ignores functionality
- Class and object models
  - Identify entities in domain and map those to objects/classes/attributes…
  - Object: Individual entity which has an identity and does not depend on another entity
  - Class: Set of objects of the same kind. Described by structure, ways of manipulation and behavious.
  - Abstract object: Abstract representation of individual object or set of objects of the same type.
  - What can be modelled in class/object models? F2.85!
  - Well suited for describing system structure. Supports data locality and property encapsulation. System decomposition can be modelled. But: Ignores funktionality and behaviour, Deep nesting not possible within one UML diagram.
- Component models

2.5.2 Behaviour Modelling

Goal: describe dynamic system behavious (How the system reacts to external events, how independent components coordinate)

Means: Harel statecharts, UML state machines, sequence diagrams.

- Statecharts
  - Models dynamic behavious.
  - + Global view of system behavious
  - + Precise and still readable
  - – Weak for modelling functionality and data
- Sequence diagrams
  - Models lifelines of components and messages between them
  - + Visualize component collaboration
  - – Confined to description of required scenarios
  - – Design-oriented, can detract from modelling requirements

2.5.3 Function and Flow Modelling

- Activity model: UML activity diagram, models process, control and data flow.
- Data and information flow: Models system functionality with data flow diagrams. Easy to understand, supports

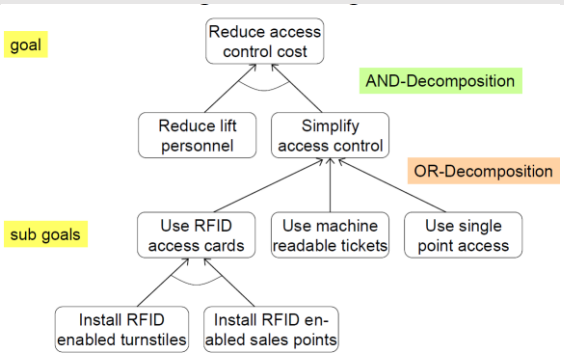
system decomposition, but no types, encapsulation etc (outdated)
- Process and workflow model

2.5.4 User-system interaction Modelling

How can a user interact with the system?

- Use case: Description of possible interactions between actors and system that, when executed, provide value.
  - From users perspective
  - Use case diagrams provide overview
  - Use case descriptions provide details
- Scenario: Description of potential sequence of events that lead to desired/unwanted result OR ordered sequence of interactions between system- and external actors.
- UML Use case diagram
  - + Provides abstract overview from actor's perspective, -ignores functions/data, -can't model hierarchy and dependencies

2.5.5 Modelling Goals



2.6 Formal Specification Languages

2.6.1 Model based formal specification

2.6.2 Z

Typical model-based formal languag. Set-based with sets, types, axioms and schemata.

- Axioms define global ariables and props
- Schemata: Namespace name, declaration and predicate.
- Relations and functions: Set of ordered tuples
- State change through operations

Example see F02.127.

2.6.3 OCL

Specification of invariants on UML models, specification on semantics of operations.

OCL expression may be part of UML model (implicit context) or written seperately (explicit context).

Summary of OCL constructs: F02.137.

TODO Excercises.

2.6.4 Proving properties

- Classic proofs (theorem proving software): is a property inferrable from set of logical statements?
  - Hard and labour-intensive
- Model checking: Explore full state space of model, prove that property holds in every possible state.
  - Fully automatic, but exploring full state space is often infeasible

TODO Minio Exercise F2.145

2.6.5 Benefits and limitations, practical use

- +Unambiguous by definition
- +Fully verifiable
- +Properties can be proven and tested automatically
- -Cost vs Value
- - Stakeholders cannot read Specs
- -Primarily for functional requirements

2.7 Validation Requirements

Identify requirements that are inadequate, incomplete/missing, inconsistent or wrong.

Also look for reqs: Not verifiable, unnecessary, not tracable, premature design decisions.

Validation: Have all stakeholders found consent on all reqs? Validate repeatedly.

2.7.1 Requirements validation techniques

- Review: author guides experts through specs, experts check those.
- Acceptance test cases
- Simulation/Animation
- Prototyping
- Formal verification/Model Checking

2.7.2 Conflict Analysis

Typical underlying reasons are:

- Misinterpretation
- Subject matter conflict (divergent needs)
- Conflict of interest (divergent interests)
- Conflict of value
- Relationship conflict (personal relationships between stakeholders)
- Organizational conflict (different hierarchy levels in organization)

Conflict resolution: Win-win-techniques (Agreement, Compromise), Win-lose-techniques (overruling, voting, prioritizing stakeholders)

2.7.3 Acceptance Testing

- Acceptance: The process of assessing whether a system satisfies all its requirements
- Acceptance test: A test that assesses whether a system satisfies all its requirements

Req Eng and Acceptance testing are intertwined. For every req and function at least one acceptance test case, for scenarios all branches and actions should be covered.

2.8 Innovate requirements

Kano's model helps identify what is implicitly expected (dissatisfiers) and explicitly required (satisfiers).

2.9 Requirements management

Attach unique identifier and metadata (author, date created/modified, source, status…) to each req.

2.9.1 Prioritizing requirements

Possible criteria: Necessity (e.g. critical, major, minor), cost of implementation, time to implement, risk, volatility.

Prio techniques: Simple ranking (Stakeholders rank reqs), Assigning points.

2.9.2 Traceability

Ability to trace a requirement back to its origins, forward to its implementation in design and code and to depending requirements.

3 Enablers and Stumble Blocks

3.1 Requirement tools

Can support: Elicitation, Documentation, Modeling, Management, Validation.

3.2 RE under time pressure

Risk-oriented spec, don't specify in uniform depth (only risky parts in full detail), employ incremental processes, don't strive for perfection.

What makes it harder: Complex domain, team is unfamiliar with domain, many stakeholders, distributed development, long cycle time, safety-critical reqs, high project risks.

Hints for Examination: Learning goals

Students have acquired basic knowledge and skills in the core methods, languages, processes, and practices in Requirements Engineering (RE). In particular, they acquired the following skills

They can
- name and describe the terms and concepts of RE
- identify stakeholders of the RE process and system boundaries
- analyse the context of a system
- differentiate between analysis activities and design activities
- evaluate risk and value of RE efforts
- classify types of requirements
- elicit requirements and document them in different forms (natural language, static models, behaviour models, user interaction models, goal models)
- select and instantiate RE processes for a given software project

In addition, they know and understand
- the methods to validate requirements
- the methods to manage requirements

Hints for the Examination

- Know main messages from the three elicitation papers
- Know UML models from chapter 9
  - Know when to use which model
  - Including use case with structured text (p. 105)
  - Additionally: AND/OR trees for goal modeling
  - Others not needed in detail, just what they are used for
- Be able to formulate use cases as in mini exercise
- Know Z and OCL basics
- Be able to structure the knowledge
  - E.g. 7 principles, three modelling views, four facets of classification
- In general: Be able to solve mini exercises
  - Except circular metro line

TODO: Conclusion