Softwaretechnik / Software-Engineering

Lecture 6: Requirements Engineering

2018-05-07

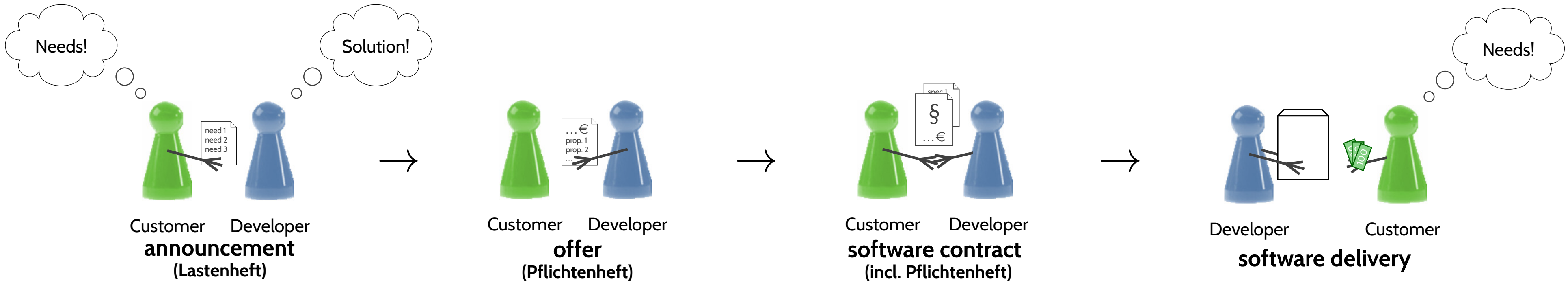
Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

You Are Here.

Introduction	L 1:	16.4., Mon
Scales, Metrics,	L 2:	19.4., Thu
Costs,	L 3:	23.4., Mon
	T 1:	26.4., Thu
Development	L 4:	30.4., Mon
Process	L 5:	3.5., Thu
Requirements	L 6:	7.5., Mon
	-	10.5., Thu
Engineering	L 7:	14.5., Mon
	T 2:	17.5., Thu
	-	21.5., Mon
	-	24.5., Thu
	L 8:	28.5., Mon
	-	31.5., Thu
	L 9:	4.6., Mon
	T 3:	7.6., Thu
	L10:	11.6., Mon
Arch. & Design,	L 11:	14.6., Thu
Software-	L12:	18.6., Mon
	T 4:	21.6., Thu
Modelling,	L13:	25.6., Mon
Patterns	L14:	28.6., Thu
QA	L15:	2.7., Mon
	T 5:	5.7., Thu
(Testing, Formal	L16:	9.7., Mon
Verification)	L17:	12.7., Thu
Wrap-Up	L18:	16.7., Mon
	T 6:	19.7., Thu

Introduction



requirement –

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

(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.

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

IEEE 610.12 (1990)

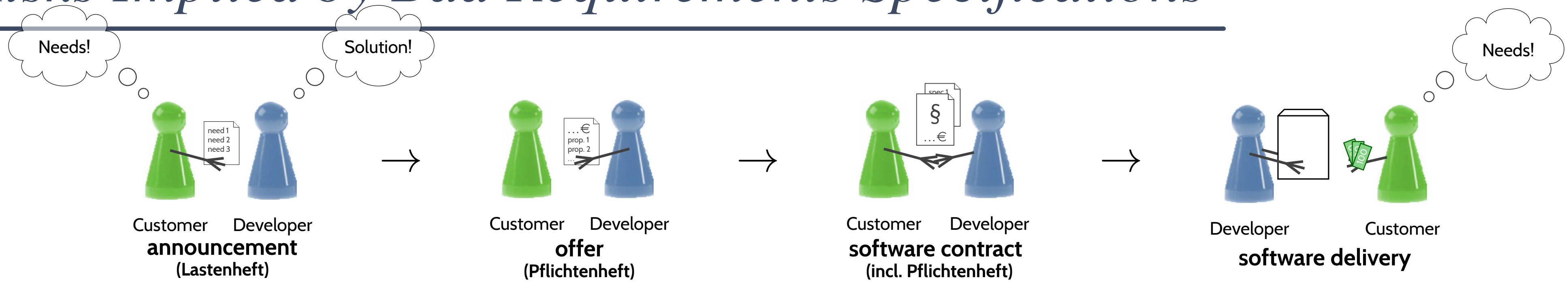
requirements analysis –

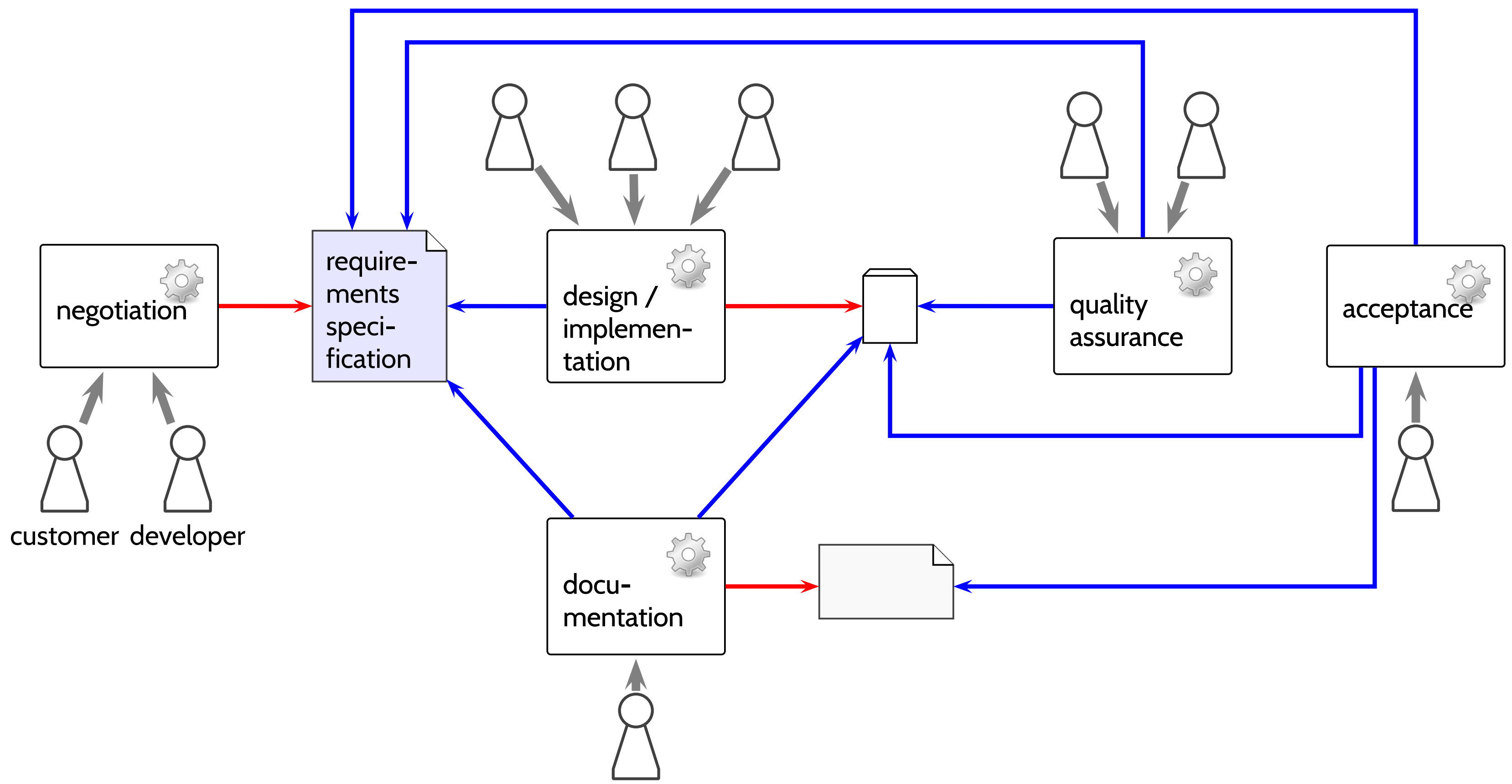
(1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements.

(2) The process of studying and refining system, hardware, or software requirements.

IEEE 610.12 (1990)

Risks Implied by Bad Requirements Specifications





Risks Implied by Bad Requirements Specifications

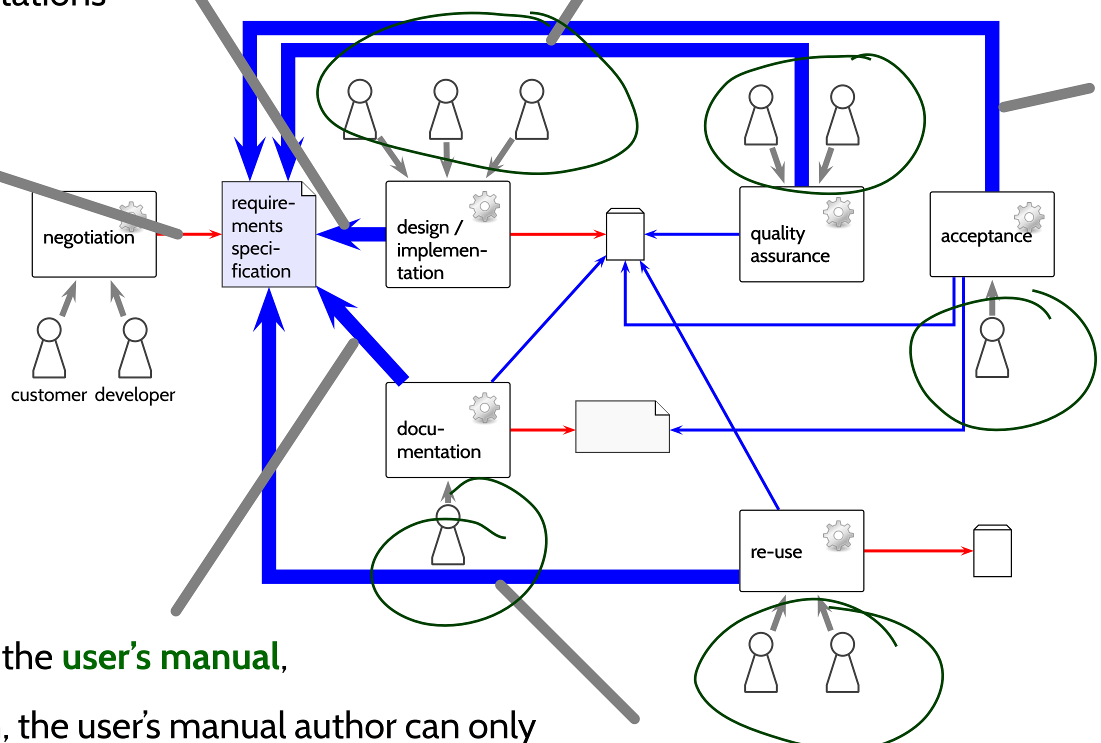
preparation of **tests**,

- without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → **systematic testing hardly possible**

design and **implementation**,

- without specification, programmers may just "ask around" when in doubt, possibly yielding different interpretations → **difficult integration**

negotiation (with customer, marketing department, or …)



acceptance by customer, **resolving** later objections or regress claims,

- without specification, it is unclear at delivery time whether behaviour is an error (developer needs to fix) or correct (customer needs to accept and pay) → **nasty disputes**, **additional effort**

documentation, e.g., the **user's manual**,

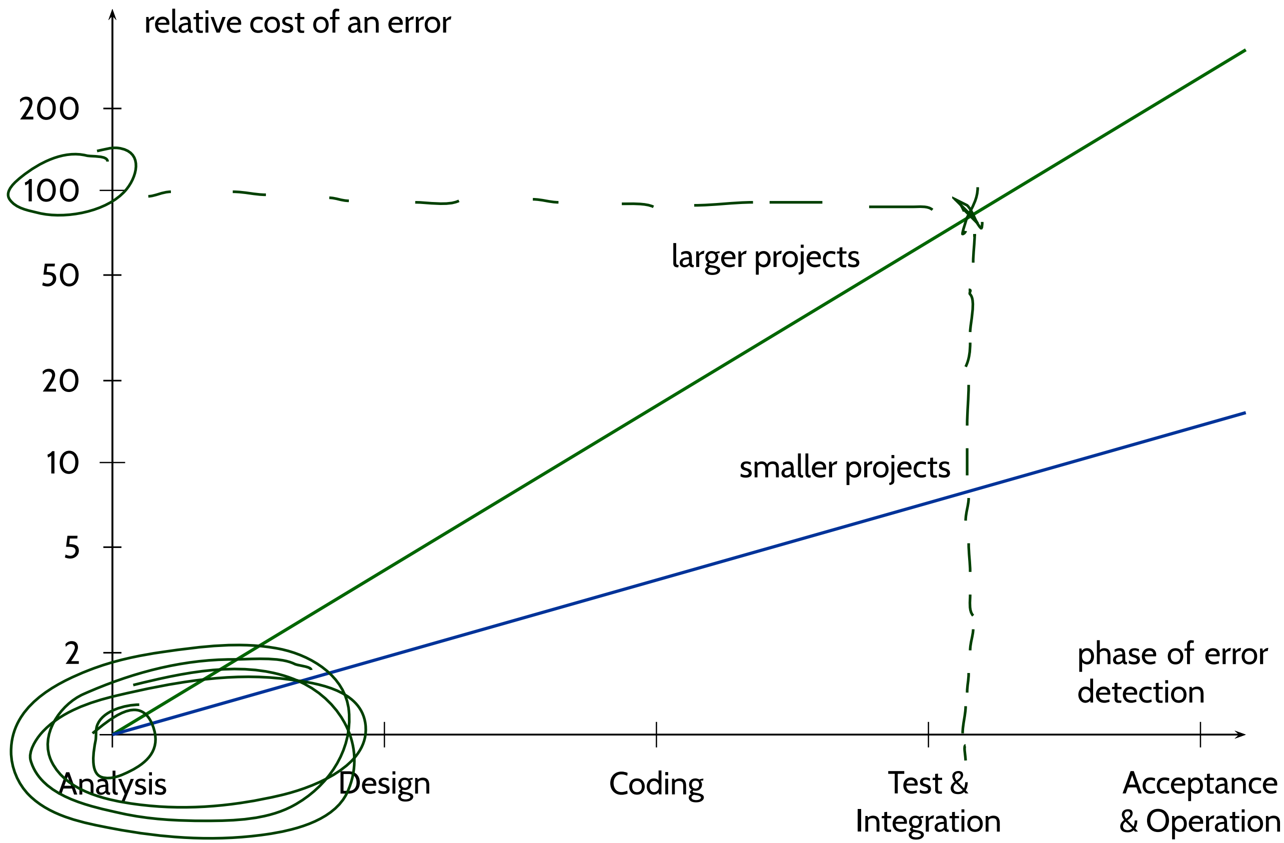
- without specification, the user's manual author can only describe what the system **does**, not what it should do (**"every observation is a feature"**)

re-use,

- without specification, re-use needs to be based on re-reading the code → **risk of unexpected changes**

- later **re-implementations**.
  - the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → **additional effort**

Discovering Fundamental Errors Late Can Be Expensive



Relative error costs over latency according to investigations at IBM, etc.

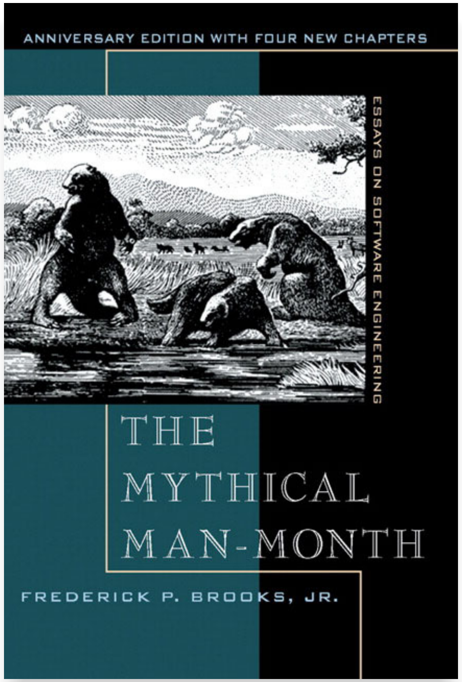
By (Boehm, 1979); Visualisation: Ludewig and Lichter (2013).

The hardest single part of building a software system is deciding precisely what to build.

No other part of the conceptual work is as difficult as establishing the detailed technical requirements ...

No other part of the work so cripples the resulting system if done wrong.

No other part is as difficult to rectify later. F.P. Brooks (Brooks, 1995)



Topic Area Requirements Engineering: Content

VL 6

- **Introduction**
- **Requirements Specification**
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary, Specification

vocabulary, concepts

- **Specification Languages**
  - Natural Language *informal*
  - Decision Tables
    - Syntax, Semantics
    - Completeness, Consistency, …

    formal

VL 7

  - Scenarios
    - User Stories, Use Cases *semi-formal*
    - Live Sequence Charts
      - Syntax, Semantics

    formal

VL 8

VL 9

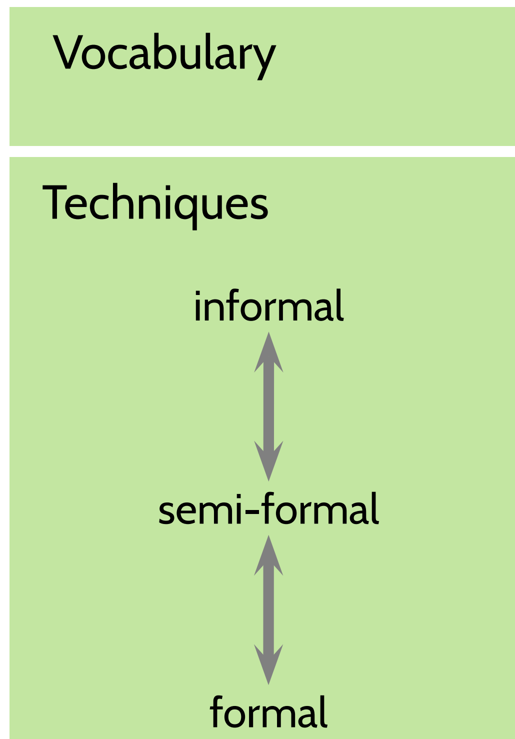
- Definition: **Software** & **SW Specification**

VL 10

- **Wrap-Up**

Recall: Structure of Topic Areas

Example: Requirements Engineering



e.g. consistent, complete, tacit, etc.

Content

- **Introduction**
  - Vocabulary: Requirements (Analysis)
  - Importance of Requirements Specifications
- **Requirements Specification**
  - Requirements Analysis
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary
  - Specification
- **Requirements Specification Languages**
  - Natural Language

Requirements Specifications

Requirements Analysis...

... in the sense of **"finding out what the exact requirements are"**.

"Analysing an existing requirements/feature specification" → later.

In the following we shall discuss:

(i) desired **properties** of
- requirements specifications,
- requirements specification documents,

(ii) **kinds** of requirements
- hard and soft,
- open and tacit,
- functional and non-functional.

(iii) (a selection of) **analysis techniques**

(iv) **documents** of the requirements analysis:
- dictionary,
- requirements specification ('Lastenheft'),
- feature specification ('Pflichtenheft').

- **Note**: In the following (unless otherwise noted), we discuss the **feature specification**, i.e. the document on which the software development is based.

  To maximise confusion, we may occasionally (inconsistently) call it **requirements specification** or just **specification** – should be clear from context...

- **Recall**: one and the same content can serve both purposes; only the title defines the purpose then.

Requirements on Requirements Specifications

A **requirements specification** should be

- **correct**
  – it correctly represents the wishes/needs of the customer,
- **complete**
  – all requirements (existing in somebody's head, or a document, or …) should be present,
- **relevant**
  – things which are not relevant to the project should not be constrained,
- **consistent**, **free of contradictions**
  – each requirement is compatible with all other requirements; otherwise the requirements are **not realisable**,
- **neutral**, **abstract**
  – a requirements specification does not constrain the realisation more than necessary,
- **traceable**, **comprehensible**
  – the sources of requirements are documented, requirements are uniquely identifiable,
- **testable**, **objective**
  – the final product can **objectively** be checked for satisfying a requirement.

- **Correctness** and **completeness** are defined **relative** to something which is usually only **in the customer's head**.

  → is is **difficult** to **be sure of correctness** and **completeness**.

- **"Dear customer, please tell me what is in your head!"** is in almost all cases **not a solution**!

  It's not unusual that even the customer does not precisely know…!

  For example, the customer may not be aware of contradictions due to technical limitations.

Requirements on Requirements Specifications

A **requirements specification** should be

- **correct**
  – it correctly represents the wishes/needs of the customer,
- **complete**
  – all requirements (existing in somebody's head, or a document, or ...) should be present,
- **relevant**
  – things which are not relevant to the project should not be constrained,
- **consistent**, free of contradictions
  – each requirement is compatible with all other requirements; otherwise the requirements are **not realisable**,
- **neutral**, **abstract**
  – a requirements specification does not constrain the realisation more than necessary,
- **traceable**, **comprehensible**
  – the sources of requirements are documented, requirements are uniquely identifiable,
- **testable**, **objective**
  – the final product can **objectively** be checked for satisfying a requirement.

- **Correctness** and **completeness** are defined **relative** to something which is usually only **in the customer's head**.

  → is is **difficult** to **be sure of correctness** and **completeness**.

- **"Dear customer, please tell me what is in your head!"** is in almost all cases **not a solution!**

  It's not unusual that even the customer does not precisely know...!

  For example, the customer may not be aware of contradictions due to technical limitations.

Excursion: Informal vs. Formal Techniques

Example: Requirements Engineering, Airbag Controller

Requirement:

Whenever a crash is detected, the airbag has to be fired within $300\,\mathrm{ms}$ $(\pm\varepsilon)$.

vs.

- Fix observables: **crashdetected** : Time → $\{0,1\}$ and **fireairbag** : Time → $\{0,1\}$
- Formalise requirement:

$$\forall\, t, t' \in \text{Time} \bullet \text{crashdetected}(t) \wedge \text{airbagfired}(t') \implies t' \in [t + 300 - \varepsilon, t + 300 + \varepsilon]$$

→ no more misunderstandings, sometimes **tools** can **objectively** decide: requirement satisfied yes/no.

Requirements on Requirements Specification Documents

The **representation** and **form** of a requirements specification should be:

- **easily understandable**,
  not unnecessarily complicated –
  all affected people should be able to understand the requirements specification,
- **precise** –
  the requirements specification should not introduce new unclarities or rooms for interpretation (→ testable, objective),
- **easily maintainable** –
  creating and maintaining the requirements specification should be easy and should not need unnecessary effort,
- **easily usable** –
  storage of and access to the requirements specification should not need significant effort.

Note: Once again, it's about compromises.

- A very precise **objective** requirements specification may not be easily understandable by every affected person.

  → provide redundant explanations.

- It is not trivial to have both, low maintenance effort and low access effort.

  → **value low access effort higher**,
  a requirements specification document is much more often **read** than **changed** or **written** (and most changes require reading beforehand).

Pitfall: Vagueness vs. Abstraction

Consider the following examples:

- **Vague** (not precise):

  "the list of participants should be sorted conveniently"
- **Precise**, abstract:

  "the list of participants should be sorted by immatriculation number, lowest number first"
- **Precise**, non-abstract:

  "the list of participants should be sorted by

```
public static <T> void Collections::sort( List<T> list, Comparator c );
```

  where T is the type of participant records, c compares immatriculation number numerically."

- A requirements specification should always be as **precise** as possible ($\rightarrow$ testable, objective).

  It need not denote **exactly one solution**;
  precisely characterising acceptable solutions is often more appropriate.
- Being too specific, may limit the design decisions of the developers, which may cause unnecessary costs.

- Idealised views advocate a strict **separation** between
  requirements ("what is to be done?") and **design** ("how are things to be done?").

Content

- **Introduction**
  - Vocabulary: Requirements (Analysis)
  - Importance of Requirements Specifications
- **Requirements Specification**
  - Requirements Analysis
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary
  - Specification
- **Requirements Specification Languages**
  - Natural Language

Kinds of Requirements

Kinds of Requirements: Functional and Non-Functional

- **Proposal**: View software S as a **function**

$$S : i_1, i_2, i_3, \ldots \mapsto o_0, o_1, o_2, \ldots$$

which maps **sequences of inputs** to **sequences of outputs**.

$\sqrt{\cdot}$: $4 \mapsto 2$
$2 \mapsto 1.41..$
$1.1 \mapsto 1..$
$\vdots$

Kinds of Requirements: Functional and Non-Functional

- **Proposal**: View software S as a **function**

$$S : i_1, i_2, i_3, \ldots \mapsto o_0, o_1, o_2, \ldots$$

which maps **sequences of inputs** to **sequences of outputs**.

Examples:

- Software "compute shipping costs":
  - o_0: initial state,
  - i_1: shipping parameters (weight, size, destination, ...),
  - o_1: shipping costs

  And no more inputs, $S : i_1 \mapsto o_1$.

- Software "traffic lights controller":
  - o_0: initial state,
  - i_1: pedestrian presses button,
  - $o_1, o_2, \ldots$: stop traffic, give green to pedestrians,
  - i_n: button pushed again
  - ...

- **Every constraint** on things which are **observable** in the sequences is a **functional requirement** (because it requires something for the function S).

  Thus **timing**, **energy consumption**, etc. may be subject to functional requirements.

- Clearly **non-functional** requirements:

  programming language, coding conventions, process model requirements, portability...

Kinds of Requirements: Hard and Soft Requirements

Kinds of Requirements: Hard and Soft Requirements

- **Example** of a **hard requirement**:
  - Cashing a cheque over N € must result in a new balance decreased by N; there is not a micro-cent of tolerance.

- **Examples** of **soft requirements**:
  - If a vending machine dispenses the selected item within 1 s, it's clearly fine; if it takes 5 min., it's clearly wrong – where's the boundary?
  - A car entertainment system which produces "noise" (due to limited bus bandwidth or CPU power) in average once per hour is acceptable, once per minute is not acceptable.

The border between hard/soft **is difficult to draw**, and

- as **developer**, we want requirements specifications to be "**as hard as possible**", i.e. we want a clear right/wrong.
- as **customer**, we often cannot provide this clarity; we know what is "**clearly wrong**" and we know what is "**clearly right**", but we don't have a sharp boundary.

$\rightarrow$ intervals, rates, etc. can serve as **precise specifications** of **soft requirements**.

Kinds of Requirements: Open and Tacit

- **open**: customer is aware of and able to explicitly communicate the requirement,

- **(semi-)tacit**:
  customer not aware of something **being** a requirement (obvious to the customer but not considered relevant by the customer, not known to be relevant).

  Examples:

  - buttons and screen of a mobile phone should be on the same side,
  - important web-shop items should be on the right hand side because the main users are socialised with right-to-left reading direction,
  - the ECU (embedded control unit) may only be allowed use a certain amount of bus capacity.

Customer/Client	Analyst: knows domain	Analyst: new to domain
explicit	requirements discovered	requirements discoverable
semi-tacit	requirements discoverable	requirements discoverable with difficulties
tacit	hard/impossible to discover	

(Gacitua et al., 2009)

- distinguish **don't care**:

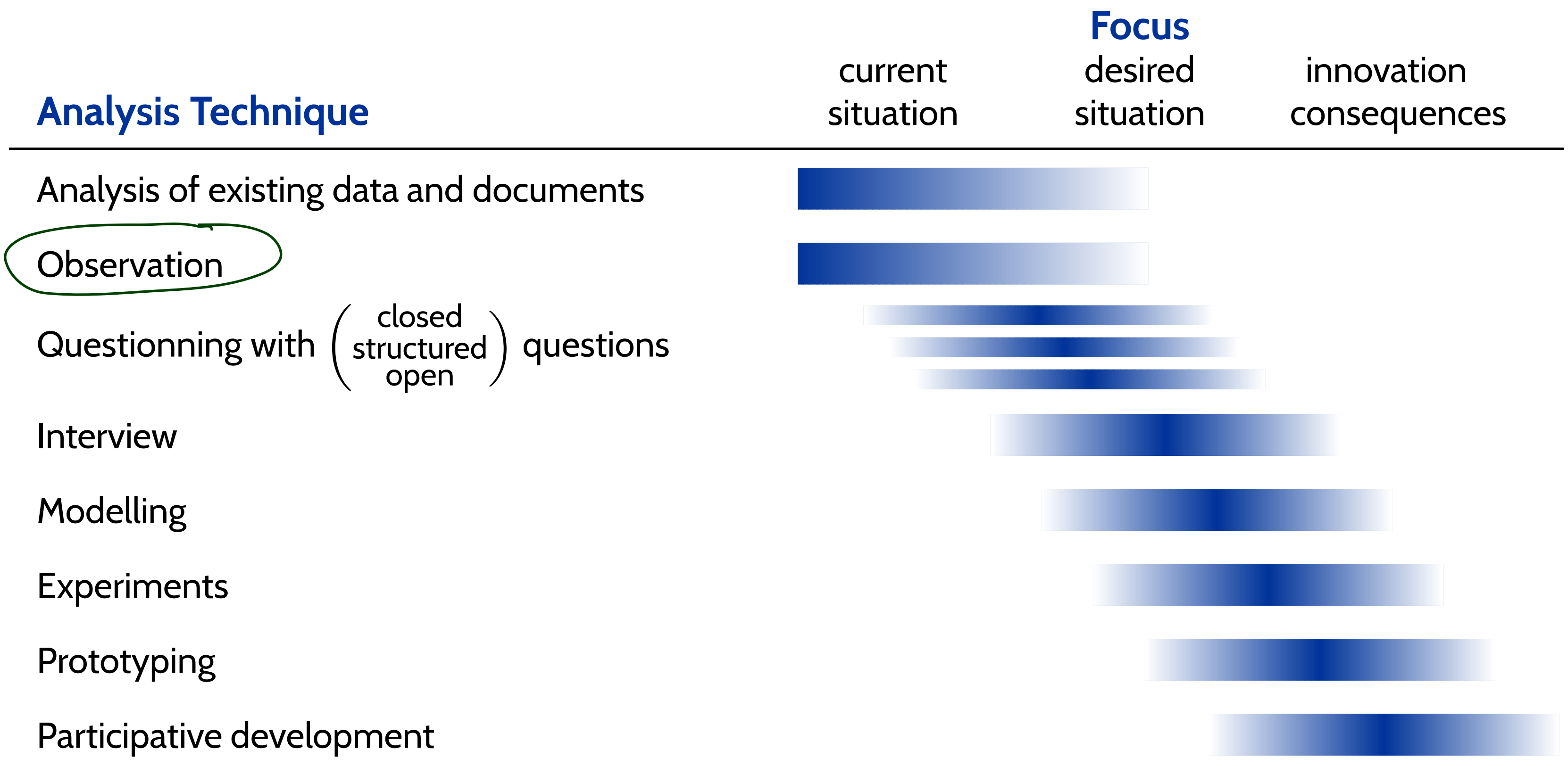
  intentionally left open to be decided by developer.

Content

- **Introduction**
  - Vocabulary: Requirements (Analysis)
  - Importance of Requirements Specifications
- **Requirements Specification**
  - Requirements Analysis
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary
  - Specification
- **Requirements Specification Languages**
  - Natural Language

Requirements Analysis Techniques

(A Selection of) Analysis Techniques



(Ludewig and Lichter, 2013)

Requirements Elicitation

- **Observation**:

  Customers can not be assumed to be trained in stating/communicating requirements.

- It is the **task of the analyst** to:
  - **ask** what is wanted,
    ask what is not wanted,
  - establish **precision**,
    look out for contradictions,
  - **anticipate** exceptions, difficulties, corner-cases,
  - have technical background to **know** technical difficulties,
  - **communicate** (formal) specification to customer,
  - "test" own understanding by **asking more** questions.

  → i.e. to **elicit** the requirements.

Goal: automate opening/closing of a main door with a new software.

A **made up** dialogue…

Analyst: *So in the morning, you open the door at the main entrance?*

  Customer: *Yes, as I told you.*

A: *Every morning?*

  C: *Of course.*

A: *Also on the weekends?*

  C: *No, on weekends, the entrance stays closed.*

A: *And during company holidays?*

  C: *Then it also remains closed of course.*

A: *And if you are ill or on vacation?*

  C: *Then Mr. M opens the door.*

A: *And if Mr. M is not available, too?*

  C: *Then the first client will knock on the window.*

A: *Okay. Now what exactly does "morning" mean?*

  …

(Ludewig and Lichter, 2013)

How Can Requirements Engineering Look In Practice?

- Set up a **core team** for analysis (3 to 4 people), include experts from the **domain** and **developers**. Analysis benefits from **highest skills** and **strong experience**.

- During analysis, talk to **decision makers** (managers), domain **experts**, and **users**.

  Users can be interviewed by a team of 2 analysts, ca. 90 min.

- The resulting "**raw material**" is sorted and assessed in half- or full-day workshops in a team of 6-10 people.

  Search for, e.g., **contradictions** between customer wishes, and for **priorisation**.

  Note: **The customer decides**. Analysts may make **proposals** (different options to choose from), but the customer chooses. (And the choice is documented.)

- The "raw material" is basis of a **preliminary requirements specification** (audience: the developers) with open questions.

- Analysts need to **communicate** the requirements specification **appropriately** (explain, give examples, point out particular corner-cases).

  Customers without strong maths/computer science background are often **overstrained** when "left alone" with a **formal** requirements specification.

- **Result**: **dictionary**, **specified requirements**.

- Many customers do not want **(radical) change**, but **improvement**.
- Good questions: How are things done today? What should be improved?

Content

- **Introduction**
  - Vocabulary: Requirements (Analysis)
  - Importance of Requirements Specifications
- **Requirements Specification**
  - Requirements Analysis
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary
  - Specification
- **Requirements Specification Languages**
  - Natural Language

Tell Them What You've Told Them...

- **Requirements Documents** are **important** – e.g., for
  - negotiation, design & implementation, documentation, testing, delivery, re-use, re-implementation.
- A **Requirements Specification** should be
  - correct, complete, relevant, consistent, neutral, traceable, objective.

  Note: vague vs. abstract.
- **Requirements Representations** should be
  - easily understandable, precise, easily maintainable, easily usable
- **Distinguish**
  - hard / soft,
  - functional / non-functional,
  - open / tacit.
- It is the task of the **analyst** to **elicit** requirements.
- Natural language is inherently imprecise, counter-measures:
  - natural language patterns.
- Do not underestimate the value of a good **dictionary**.

References

References

Arenis, S. F., Westphal, B., Dietsch, D., Muñiz, M., and Andisha, A. S. (2014). The wireless fire alarm system: Ensuring conformance to industrial standards through formal verification. In Jones, C. B., Pihlajasaari, P., and Sun, J., editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *LNCS*, pages 658–672. Springer.

Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications. In *EURO IFIP 79*, pages 711–719. Elsevier North-Holland.

Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley.

Gacitua, R., Ma, L., Nuseibeh, B., Piwek, P., de Roeck, A., Rouncefield, M., Sawyer, P., Willis, A., and Yang, H. (2009). Making tacit requirements explicit. talk.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*. Std 830-1998.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rupp, C. and die SOPHISTen (2009). *Requirements-Engineering und -Management*. Hanser, 5th edition.