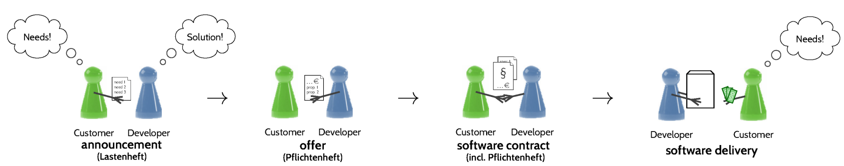
Softwaretechnik / Software-Engineering

Lecture 7: Formal Methods for Requirements Engineering

2018-05-14

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

Albert-Ludwigs-Universität Freiburg, Germany



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

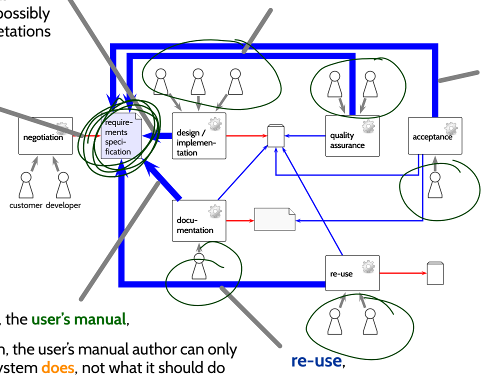
design and **implementation**,

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

preparation of **tests**,

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

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**

Topic Area Requirements Engineering: Content

VL 6

- **Introduction**
- **Requirements Specification**
  - Desired Properties
  - Kinds of Requirements
  - Analysis Techniques
- **Documents**
  - Dictionary, Specification
- **Specification Languages**
  - Natural Language
  - Decision Tables (VL 7)
    - Syntax, Semantics
    - Completeness, Consistency, …
  - Scenarios (VL 8)
    - User Stories, Use Cases
    - Live Sequence Charts (VL 9)
      - Syntax, Semantics
- Definition: **Software** & **SW Specification** (VL 10)
- **Wrap-Up**

Content

- **Documents**
  - Dictionary
  - Specification
- **Requirements Specification Languages**
  - Natural Language

Content

- **(Basic) Decision Tables**
  - Syntax, Semantics
- **...for Requirements Specification**
- **...for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
- **Discussion**

Logic

Requirements Documents

IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Recommended Practice for Software Requirements Specifications

Sponsor

Software Engineering Standards Committee
of the
IEEE Computer Society

Approved 25 June 1998

IEEE-SA Standards Board

Abstract: The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.
Keywords: contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 0-7381-0332-2

Structure of a Requirements Document: Example

1 INTRODUCTION
- 1.1 Purpose
- 1.2 Acronyms and Definitions
- 1.3 References
- 1.4 User Characteristics

2 FUNCTIONAL REQUIREMENTS
- 2.1 Function Set 1
- 2.2 etc.

3 REQUIREMENTS TO EXTERNAL INTERFACES
- 3.1 User Interfaces
- 3.2 Interfaces to Hardware
- 3.3 Interfaces to Software Products / Software / Firmware
- 3.4 Communication Interfaces

4 REQUIREMENTS REGARDING TECHNICAL DATA
- 4.1 Volume Requirements
- 4.2 Performance
- 4.3 etc.

5 GENERAL CONSTRAINTS AND REQUIREMENTS
- 5.1 Standards and Regulations
- 5.2 Strategic Constraints
- 5.3 Hardware
- 5.4 Software
- 5.5 Compatibility
- 5.6 Cost Constraints
- 5.7 Time Constraints
- 5.8 etc.

6 PRODUCT QUALITY REQUIREMENTS
- 6.1 Availability, Reliability, Robustness
- 6.2 Security
- 6.3 Maintainability
- 6.4 Portability
- 6.5 etc.

7 FURTHER REQUIREMENTS
- 7.1 System Operation
- 7.2 Customisation
- 7.3 Requirements of Internal Users

(Ludewig and Lichter, 2013) based on (IEEE, 1998)

Example Requirement

The loss of the ability of the system to transmit a signal from a component to the central unit is
- detected in less than 300 seconds and
- displayed at the central unit within 100 seconds thereafter.

Dictionary

- Requirements analysis should be based on a **dictionary**.
- A **dictionary** comprises definitions and clarifications of **terms** that are relevant to the project and of which different people (in particular customer and developer) may have different understandings before agreeing on the dictionary.
- Each **entry** in the **dictionary** should provide the following information:
  - **term** and **synonyms** (in the sense of the requirements specification),
  - **meaning** (definition, explanation),
  - **deliminations** (where **not** to use this terms),
  - **validness** (in time, in space, ...),
  - **denotation**, unique identifiers, ...,
  - **open questions** not yet resolved,
  - **related terms**, cross references.

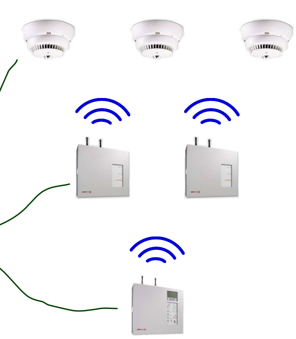
  Note: entries for terms that **seemed** "crystal clear" at first sight are **not uncommon**.
- All work on requirements should, as far as possible, be done **using terms from the dictionary** consistently and consequently.

  The dictionary should in particular be **negotiated with the customer** and used in communication (if not possible, at least developers should stick to dictionary terms).
- **Note**: do not mix up **real-world/domain** terms with ones only "living" in the software.

Dictionary Example

Example: **Wireless Fire Alarm System**

- During a project on designing a highly reliable, EN-54-25 conforming wireless communication protocol, we had to learn that the relevant components of a fire alarm system are
  - **terminal participants** (heat/smoke sensors and manual indicators),
  - **repeaters** (a non-terminal participant),
  - and **a central unit** (not a participant).
- Repeaters and central unit are technically very similar, but need to be distinguished to understand requirements. The **dictionary** explains these terms.



(Arenis et al., 2014)

Excerpt from the dictionary (ca. 50 entries in total):

Part A part of a fire alarm system is either a **participant** or a **central unit**.

Repeater A repeater is a **participant** which accepts messages for the **central unit** from other **participants**, or messages from the **central unit** to other **participants**.

Central Unit A central unit is a **part** which receives messages from different assigned **participants**, assesses the messages, and reacts, e.g. by forwarding to persons or optical/acustic signalling devices.

Terminal Participant A terminal participant is a **participant** which is not a **repeater**. Each terminal participant consists of exactly one wireless communication module and devices which provide sensor and/or signalling functionality.

Back to the Example Requirement

The loss of the ability of the system to transmit a signal from a component to the central unit is

- detected in less than 300 seconds and
- displayed at the central unit within 100 seconds thereafter.

Requirements Specification

specification – A document that specifies,

- in a complete, precise, verifiable manner,

the

- requirements, design, behavior, or other characteristics of a system or component,

and, often, the procedures for determining whether these provisions have been satisfied. **IEEE 610.12 (1990)**

software requirements specification (SRS) – Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces. **IEEE 610.12 (1990)**

Content

- **(Basic) Decision Tables**
  - Syntax, Semantics
- **. . . for Requirements Specification**
- **. . . for Requirements Analysis**
  - Completeness,
  - Useless Rules,
  - Determinism
- **Domain Modelling**
  - Conflict Axiom,
  - Relative Completeness,
  - Vacuous Rules,
  - Conflict Relation
- **Collecting Semantics**
- **Discussion**

Logic

Requirements Specification Languages

Requirements Specification Language

specification language – A language, often a machine-processible combination of natural and formal language, used to express the requirements, design, behavior, or other characteristics of a system or component.

For example, a design language or requirements specification language. Contrast with: programming language; query language. **IEEE 610.12 (1990)**

requirements specification language – A specification language with special constructs and, sometimes, verification protocols, used to develop, analyze, and document hardware or software requirements. **IEEE 610.12 (1990)**

Natural Language Specification *(Ludewig and Lichter, 2013) based on (Rupp and die SOPHISTen, 2009)*

	rule	explanation, example
R1	State each requirement in **active voice**.	Name the actors, indicate whether the user or the system does something. Not "the item is deleted".
R2	Express processes by **full verbs**.	Not "is", "has", but "reads", "creates"; full verbs require information which describe the process more precisely. Not "when data is consistent" but "after program P has checked consistency of the data".
R3	Discover **incompletely defined verbs**.	In "the component raises an error", ask whom the message is addressed to.
R4	Discover **incomplete conditions**.	Conditions of the form "if-else" need descriptions of the if- and the then-case.
R5	Discover **universal quantifiers**.	Are sentences with "never", "always", "each", "any", "all" really universally valid? Are "all" really all or are there exceptions.
R6	Check **nominalisations**.	Nouns like "registration" often hide complex processes that need more detailed descriptions; the verb "register" raises appropriate questions: who, where, for what?
R7	Recognise and refine **unclear substantives**.	Is the substantive used as a generic term or does it denote something specific? Is "user" generic or is a member of a specific classes meant?
R8	Clarify **responsibilities**.	If the specification says that something is "possible", "impossible", or "may", "should", "must" happen, clarify who is enforcing or prohibiting the behaviour.
R9	Identify **implicit assumptions**.	Terms ("the firewall") that are not explained further often hint to implicit assumptions (here: there seems to be a firewall).

Natural Language Patterns

Natural language requirements can be (tried to be) written as an instance of the **pattern** "$\langle A\rangle\ \langle B\rangle\ \langle C\rangle\ \langle D\rangle\ \langle E\rangle\ \langle F\rangle$." (German grammar) where

A	clarifies when and under what conditions the activity takes place
B	is MUST (obligation), SHOULD (wish), or WILL (intention); also: MUST NOT (forbidden)
C	is either "the system" or the concrete name of a (sub-)system
D	one of three possibilities: • "does", description of a system activity, • "offers", description of a function offered by the system to somebody, • "is able if", usage of a function offered by a third party, under certain conditions
E	extensions, in particular an object
F	the actual process word (what happens)

(Rupp and die SOPHISTen, 2009)

Example:

After office hours ($= A$), the system ($= C$) should ($= B$) offer to the operator ($= D$) a backup ($= F$) of all new registrations to an external medium ($= E$).

Other Pattern Example: RFC 2119

```
Network Working Group                                         S. Bradner
Request for Comments: 2119                            Harvard University
BCP: 14                                                       March 1997
Category: Best Current Practice


        Key words for use in RFCs to Indicate Requirement Levels

Status of this Memo

   This document specifies an Internet Best Current Practices for the
   Internet Community, and requests discussion and suggestions for
   improvements.  Distribution of this memo is unlimited.

Abstract

   In many standards track documents several words are used to signify
   the requirements in the specification.  These words are often
   capitalized.  This document defines these words as they should be
   interpreted in IETF documents.  Authors who follow these guidelines
   should incorporate this phrase near the beginning of their document:

      The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
      NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED",  "MAY", and
      "OPTIONAL" in this document are to be interpreted as described in
      RFC 2119.

   Note that the force of these words is modified by the requirement
   level of the document in which they are used.

1. MUST   This word, or the terms "REQUIRED" or "SHALL", mean that the
   definition is an absolute requirement of the specification.

2. MUST NOT   This phrase, or the phrase "SHALL NOT", mean that the
   definition is an absolute prohibition of the specification.

3. SHOULD   This word, or the adjective "RECOMMENDED", mean that there
   may exist valid reasons in particular circumstances to ignore a
   particular item, but the full implications must be understood and
   carefully weighed before choosing a different course.

4. SHOULD NOT   This phrase, or the phrase "NOT RECOMMENDED" mean that
   there may exist valid reasons in particular circumstances when the
   particular behavior is acceptable or even useful, but the full
   implications should be understood and the case carefully weighed
   before implementing any behavior described with this label.



Bradner                      Best Current Practice                [Page 1]
```

```
RFC 2119                     RFC Key Words

5. MAY   This word, or the adjective "OPTIONAL", mean that a
   truly optional.  One vendor may choose to include the ite
   particular marketplace requires it or because the vendor
   it enhances the product while another vendor may omit the
   An implementation which does not include a particular opt
   prepared to interoperate with another implementation whic
   include the option, though perhaps with reduced functiona
   same vein an implementation which does include a particul
   MUST be prepared to interoperate with another implementat
   does not include the option (except, of course, for the f
   option provides.)

6. Guidance in the use of these Imperatives

   Imperatives of the type defined in this memo must be usec
   and sparingly.  In particular, they MUST only be used whe
   actually required for interoperation or to limit behavior
   potential for causing harm (e.g., limiting retransmisssic
   example, they must not be used to try to impose a particu
   on implementors where the method is not required for
   interoperability.

7. Security Considerations

   These terms are frequently used to specify behavior with
   implications.  The effects on security of not implementir
   SHOULD, or doing something the specification says MUST NC
   NOT be done may be very subtle. Document authors should t
   to elaborate the security implications of not following
   recommendations or requirements as most implementors will
   had the benefit of the experience and discussion that prc
   specification.

8. Acknowledgments

   The definitions of these terms are an amalgam of definiti
   from a number of RFCs.  In addition, suggestions have bee
   incorporated from a number of people including Robert Ull
   Narten, Neal McBurnett, and Robert Elz.
```

Content

Logic

Decision Tables

Decision Tables: Example

T		r_1	r_2	r_3
c_1		$\times$	$\times$	$-$
c_2		$\times$	$-$	$*$
c_3		$-$	$\times$	$*$
a_1		$\times$	$-$	$-$
a_2		$-$	$\times$	$-$

name
description (opt.)
rule
rule name
premise
conditions
don't care
actions
effect

Decision Table Syntax

- Let C be a set of **conditions** and A be a set of **actions** s.t. $C \cap A = \emptyset$.
- A **decision table** T **over** C **and** A is a labelled $(m+k) \times n$ matrix

T: decision table		r_1	$\cdots$	r_n
c_1	description of condition c_1	$v_{1,1}$	$\cdots$	$v_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_m	description of condition c_m	$v_{m,1}$	$\cdots$	$v_{m,n}$
a_1	description of action a_1	$w_{1,1}$	$\cdots$	$w_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_k	description of action a_k	$w_{k,1}$	$\cdots$	$w_{k,n}$

- where
  - $c_1, \dots, c_m \in C$,
  - $a_1, \dots, a_k \in A$,
  - $v_{1,1}, \dots, v_{m,n} \in \{-, \times, *\}$ and
  - $w_{1,1}, \dots, w_{k,n} \in \{-, \times\}$.
- Columns $(v_{1,i}, \dots, v_{m,i}, w_{1,i}, \dots, w_{k,i})$, $1 \leq i \leq n$, are called **rules**,
- $r_1, \dots, r_n$ are **rule names**.
- $(v_{1,i}, \dots, v_{m,i})$ is called **premise** of rule r_i,
  $(w_{1,i}, \dots, w_{k,i})$ is called **effect** of r_i.

Decision Table Semantics

Each rule $r \in \{r_1, \dots, r_n\}$ of table T

T: decision table		r_1	$\cdots$	r_n
c_1	description of condition c_1	$v_{1,1}$	$\cdots$	$v_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_m	description of condition c_m	$v_{m,1}$	$\cdots$	$v_{m,n}$
a_1	description of action a_1	$w_{1,1}$	$\cdots$	$w_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_k	description of action a_k	$w_{k,1}$	$\cdots$	$w_{k,n}$

is assigned to a **propositional logical formula** $\mathcal{F}(r)$ over signature $C \dot\cup A$ as follows:

- Let $(v_1, \dots, v_m)$ and $(w_1, \dots, w_k)$ be premise and effect of r.
- Then

$$\mathcal{F}(r) := \underbrace{F(v_1, c_1) \wedge \dots \wedge F(v_m, c_m)}_{=:\mathcal{F}_{pre}(r)} \wedge \underbrace{F(w_1, a_1) \wedge \dots \wedge F(w_k, a_k)}_{=:\mathcal{F}_{\mathit{eff}}(r)}$$

  where

$$F(v, x) = \begin{cases} x & \text{, if } v = \times \\ \neg x & \text{, if } v = - \\ \mathit{true} & \text{, if } v = * \end{cases}$$

Decision Table Semantics: Example

$$\mathcal{F}(r) := F(v_1, c_1) \wedge \dots \wedge F(v_m, c_m) \wedge F(v_1, a_1) \wedge \dots \wedge F(v_k, a_k)$$

$$F(v, x) = \begin{cases} x & \text{, if } v = \times \\ \neg x & \text{, if } v = - \\ \mathit{true} & \text{, if } v = * \end{cases}$$

T		r_1	r_2	r_3
c_1		$\times$	$\times$	$-$
c_2		$\times$	$-$	$*$
c_3		$-$	$\times$	$*$
a_1		$\times$	$-$	$-$
a_2		$-$	$\times$	$-$

- $\mathcal{F}(r_1) = F(\times, c_1) \wedge F(\times, c_2) \wedge F(-, c_3) \wedge F(\times, a_1) \wedge F(-, a_2)$
  $= c_1 \wedge c_2 \wedge \neg c_3 \wedge a_1 \wedge \neg a_2$
- $\mathcal{F}(r_2) = c_1 \wedge \neg c_2 \wedge c_3 \wedge \neg a_1 \wedge a_2$
- $\mathcal{F}(r_3) = \neg c_1 \wedge \mathit{true} \wedge \mathit{true} \wedge \neg a_1 \wedge \neg a_2$

Tell Them What You've Told Them...

- **Decision Tables**: **one example** for a **formal requirements specification language** with
  - formal syntax,
  - formal semantics.
- Requirements analysts can use **DTs** to
  - **formally** (objectively, precisely)

  describe **their understanding** of requirements.
  Customers may need translations/explanation!
- **DT** properties like
  - (relative) completeness, determinism,
  - uselessness,

  can be used to **analyse** requirements.

  The discussed DT properties are **decidable**,
  there can be **automatic** analysis tools.
- **Domain modelling** formalises assumptions on the context of software; for DTs:
  - conflict axioms, conflict relation,

  Note: wrong assumptions can have serious consequences.

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.

Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum, 3rd edition.

Bjørner, D. (2006). *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Springer-Verlag.

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.

Kopetz, H. (2011). What I learned from Brian. In Jones, C. B. et al., editors, *Dependable and Historic Computing*, volume 6875 of *LNCS*. Springer.

Lovins, A. B. and Lovins, L. H. (2001). *Brittle Power - Energy Strategy for National Security*. Rocky Mountain Institute.

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.

Wikipedia (2015). Lufthansa flight 2904. id 646105486, Feb., 7th, 2015.