**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

**negotiation** *(with customer, marketing department, or …)*

**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”)

**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

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

**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

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

Requirements Specification
- Desired Properties
- Kinds of Requirements
- Analysis Techniques

Documents
- Dictionary, Specification

Specification Languages
- Natural Language
- Decision Tables
  - Syntax, Semantics
  - Completeness, Consistency, ...

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

Definition: Software & SW Specification

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
Requirements Documents
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

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published 1998. Printed in the United States of America.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.
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)
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.
**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.

**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/acoustic 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.
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.
**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.

---

**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)
(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
Requirements Specification Languages
**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.

**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)*

<table>
<thead>
<tr>
<th>rule</th>
<th>explanation, example</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>State each requirement in <strong>active voice</strong>. Name the actors, indicate whether the user or the system does something. Not “the item is deleted”.</td>
</tr>
<tr>
<td>R2</td>
<td>Express processes by <strong>full verbs</strong>. 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”.</td>
</tr>
<tr>
<td>R3</td>
<td>Discover <strong>incompletely defined verbs</strong>. In &quot;the component raises an error&quot;; ask whom the message is addressed to.</td>
</tr>
<tr>
<td>R4</td>
<td>Discover <strong>incomplete conditions</strong>. Conditions of the form “if-else” need descriptions of the if- and the then-case.</td>
</tr>
<tr>
<td>R5</td>
<td>Discover <strong>universal quantifiers</strong>. Are sentences with “never”, “always”, “each”, “any”, “all” really universally valid? Are “all” really all or are there exceptions.</td>
</tr>
<tr>
<td>R6</td>
<td>Check <strong>nominalisations</strong>. Nouns like “registration” often hide complex processes that need more detailed descriptions; the verb “register” raises appropriate questions: who, where, for what?</td>
</tr>
<tr>
<td>R7</td>
<td>Recognise and refine <strong>unclear substantives</strong>. 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?</td>
</tr>
<tr>
<td>R8</td>
<td>Clarify <strong>responsibilities</strong>. If the specification says that something is “possible”, “impossible”, or “may”, “should”, “must” happen, clarify who is enforcing or prohibiting the behaviour.</td>
</tr>
<tr>
<td>R9</td>
<td>Identify <strong>implicit assumptions</strong>. Terms (“the firewall”) that are not explained further often hint to implicit assumptions (here: there seems to be a firewall).</td>
</tr>
</tbody>
</table>
Natural Language Patterns

Natural language requirements can be (tried to be) written as an instance of the pattern “⟨A⟩ ⟨B⟩ ⟨C⟩ ⟨D⟩ ⟨E⟩ ⟨F⟩.” (German grammar) where

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

(Rupp and die SOPHISTen, 2009)

Example:

After office hours (≡ A), the system (≡ C) should (≡ B) offer to the operator (≡ D) a backup (≡ E) of all new registrations to an external medium (≡ E).
Network Working Group                                         S. Bradner
Request for Comments: 2119                                         Harvard University
BCP: 14                                         March 1997
Category: Best Current Practice

5. MAY   This word, or the adjective "OPTIONAL", mean that a
          particular marketplace requires it or because the vendor
          does not include the option except, of course, for the
          option provides.

6. Guidance in the use of these Imperatives

      Imperatives of the type defined in this memo must be used
          and sparingly. In particular, they MUST only be used when
          actually required for interoperation or to limit behavioral
          potential for causing harm (e.g., limiting retransmissions).
          For example, they must not be used to try to impose a particular
          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 implementing
          SHOULD, or doing something the specification says MUST
          NOT be done may be very subtle. Document authors should
          to elaborate the security implications of not following
          recommendations or requirements as most implementors will
          had the benefit of the experience and discussion that produced
          specification.

8. Acknowledgments

      The definitions of these terms are an amalgam of definitions
          from a number of RFCs. In addition, suggestions have been
          incorporated from a number of people including Robert Ullman,
          Narten, Neal Mcburnett, and Robert Elz.
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
Decision Tables
## Decision Tables: Example

### Table

<table>
<thead>
<tr>
<th>T</th>
<th>r₁</th>
<th>r₂</th>
<th>r₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>c₂</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>c₃</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>a₁</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>a₂</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

- **Name**
- **Rule**
- **Rule Name**
- **Description (opt.)**
- **Premise**
- **Default Core**
- **Effect**
- **Conditions**
- **Actions**
Let \( C \) be a set of \textbf{conditions} and \( A \) be a set of \textbf{actions} s.t. \( C \cap A = \emptyset \).

A \textbf{decision table} \( T \) over \( C \) and \( A \) is a labelled \((m + k) \times n\) matrix

<table>
<thead>
<tr>
<th>( T: ) decision table</th>
<th>( r_1 )</th>
<th>( \cdots )</th>
<th>( r_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>description of condition ( c_1 )</td>
<td>( v_{1,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( c_m )</td>
<td>description of condition ( c_m )</td>
<td>( v_{m,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>description of action ( a_1 )</td>
<td>( w_{1,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( a_k )</td>
<td>description of action ( a_k )</td>
<td>( w_{k,1} )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

where

- \( c_1, \ldots, c_m \in C \),
- \( a_1, \ldots, a_k \in A \),
- \( v_{1,1}, \ldots, v_{m,n} \in \{-, \times, *\} \) and
- \( w_{1,1}, \ldots, w_{k,n} \in \{-, \times\} \).

- Columns \( \{v_{1,i}, \ldots, v_{m,i}, w_{1,i}, \ldots, w_{k,i}\}, 1 \leq i \leq n \), are called \textbf{rules},
- \( r_1, \ldots, r_n \) are \textbf{rule names}.
- \( \{v_{1,i}, \ldots, v_{m,i}\} \) is called \textbf{premise} of rule \( r_i \),
- \( \{w_{1,i}, \ldots, w_{k,i}\} \) is called \textbf{effect} of \( r_i \).
Decision Table Semantics

Each rule \( r \in \{ r_1, \ldots, r_n \} \) of table \( T \)

<table>
<thead>
<tr>
<th>( T ): decision table</th>
<th>( r_1 )</th>
<th>( \cdots )</th>
<th>( r_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>description of condition ( c_1 )</td>
<td>( v_{1,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( c_m )</td>
<td>description of condition ( c_m )</td>
<td>( v_{m,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>description of action ( a_1 )</td>
<td>( w_{1,1} )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( a_k )</td>
<td>description of action ( a_k )</td>
<td>( w_{k,1} )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

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

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

\[
\mathcal{F}(r) := \mathcal{F}(v_1, c_1) \land \cdots \land \mathcal{F}(v_m, c_m) \land \mathcal{F}(w_1, a_1) \land \cdots \land \mathcal{F}(w_k, a_k)
\]

where

\[
\mathcal{F}(v, x) = \begin{cases} 
  x, & \text{if } v = \times \\
  \neg x, & \text{if } v = - \\
  \text{true}, & \text{if } v = * 
\end{cases}
\]
## Decision Table Semantics: Example

\[ F(r) := F(v_1, c_1) \land \cdots \land F(v_m, c_m) \land F(v_1, a_1) \land \cdots \land F(v_k, a_k) \]

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

<table>
<thead>
<tr>
<th>( T )</th>
<th>( r_1 )</th>
<th>( r_2 )</th>
<th>( r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>\times</td>
<td>\times</td>
<td>\times</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>\times</td>
<td>\times</td>
<td>\times</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>\times</td>
<td>\times</td>
<td>\times</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>\times</td>
<td>\times</td>
<td>\times</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>\times</td>
<td>\times</td>
<td>\times</td>
</tr>
</tbody>
</table>

- \( F(r_1) = F(x, c_1) \land F(x, c_2) \land F(-, c_3) \land F(x, a_1) \land F(-, a_2) \)
  \[= c_1 \land c_2 \land \neg c_3 \land a_1 \land \neg a_2 \]
- \( F(r_2) = c_1 \land \neg c_2 \land c_3 \land \neg a_1 \land a_2 \)
- \( F(r_3) = \neg c_1 \land \text{true} \land \text{true} \land \neg a_1 \land \neg a_2 \)
... so, off to “‘technological paradise’ where [...] everything happens according to the blueprints”.

(Kopetz, 2011; Lovins and Lovins, 2001)
**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


