Risks Implied by Bad Requirements Specifications

- Negotiation (with customer, marketing department, etc.): Without specification, programmers may just "ask around" when in doubt, possibly yielding different interpretations.

- Design and implementation: Without specification, programmers may not have a clear understanding of the requirements, leading to difficulty in integration.

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

- Preparation of tests: Without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes). Systematic testing is hardly possible.

- Acceptance by customer: 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). This can lead to nasty disputes and additional effort.

- Re-use: Without specification, re-use needs to be based on re-reading the code, which can lead to additional effort and risk of unexpected changes later.

- 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.
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

- A central unit
- Repeaters
- Terminal participants

These components can consist of exactly one wireless communication module and devices which provide sensor and/or signalling functionality. They are used in communication (if not possible, developers should stick to dictionary terms).

All work on requirements should, as far as possible, be done

- Consistently and consequently.
- Using terms from the dictionary.
- Related terms.
- Cross references.
- Example from the dictionary.

The dictionary should in particular be

- Consistent and
- Consistent and
- Consistent and
- Consistent.

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

- Displayed at the central unit within 100 seconds thereafter.
- Displayed at the central unit within 100 seconds thereafter.
- Displayed at the central unit within 100 seconds thereafter.

The contract, customer, prototyping, software requirements specification, supplier, system

- Consistently and
- Consistently and
- Consistently and
- Consistently.

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

Copyright © 1998, IEEE

All rights reserved. Published 1998. Printed in the United States of America.


Requirements Documents

<table>
<thead>
<tr>
<th>Requirements Document: Example</th>
<th>Requirements Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Requirement</td>
<td>Example Requirement</td>
</tr>
<tr>
<td>Example Requirement</td>
<td>Example Requirement</td>
</tr>
</tbody>
</table>

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 in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.
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)

• Natural Language Specification (Ludewig and Lichter, 2013) based on (Rupp and die SOPHIST en, 2009)

Rules:

• (Basic) Decision Tables
• Syntax, Semantics
• 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)

• 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 class 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).
### Decision Table Syntax

#### Decision Tables: Example

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
</tbody>
</table>

#### Collecting Semantics

- "is able if" usage of a function offered by a third party, under certain conditions
- Determinism
- Completeness
- Syntax, Semantics
- Collecting Semantics

#### After office hours

- Extensions, in particular an object
- Conflict Axiom
- Vacuous Rules
- Useful Rules
- For Requirements Analysis

### Network Working Group

- RFC 2119: Request for Comments
- Best Current Practice

### RFC Key Words

- MAY: This word, or the adjective "OPTIONAL," should be used when the application of the label is optional.
- SHOULD: The label should be used in all circumstances.
- MUST: The label must be used, there is no exception.

5. MAY   This word, or the adjective "OPTIONAL," should be used when the application of the label is optional.

---

A pattern representing the actual process word (what happens) of all new registrations to an external medium should, the system, "is able if" usage of a function offered by a third party, under certain conditions.