Softwaretechnik / Software-Engineering

Lecture 6: Formal Methods for Requirements Engineering

2019-05-16

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
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

- (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
**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)
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
(Ludewig and Lichter, 2013) based on (IEEE, 1998)
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),
- **delimitations** (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

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 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.

**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.
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.

*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-Based 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 “the component raises an error”, 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>. ( \forall, \exists ) 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 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>clarifies when and under what conditions the activity takes place</td>
</tr>
<tr>
<td>( B )</td>
<td>is MUST (obligation), SHOULD (wish), or WILL (intention); also: MUST NOT (forbidden)</td>
</tr>
<tr>
<td>( C )</td>
<td>is either “the system” or the concrete name of a (sub-)system</td>
</tr>
</tbody>
</table>
| \( D \) | 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) |

(\textit{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
BCP: 14
Category: Best Current Practice

RFC 2119                     RFC Key Words
MAY

5. MAY   This word, or the adjective "OPTIONAL", mean that a
   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 option
   prepared to interoperate with another implementation which
   include the option, though perhaps with reduced functional
   same vein an implementation which does include a particular
   MUST be prepared to interoperate with another implementation
   does not include the option (except, of course, for the feature
   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 interoperability 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 try
to elaborate the security implications of not following
recommendations or requirements as most implementors will
not have had the benefit of the experience and discussion that produced
the 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 Ullmann,
Narten, Neal McBurnett, and Robert Elz.
Content

- Documents
  - Dictionary, Specification

- Requirements Specification Languages
  - Natural Language

- (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
Definition. [Bjørner and Havelund (2014)]
A method is called formal method if and only if its techniques and tools can be explained in mathematics.

Example:
If a method includes a specification language (as a tool), then that language has

- a formal syntax,
- a formal semantics, and
- a formal proof system.
"The techniques of a formal method help

- construct a specification, and/or
- analyse a specification, and/or
- transform (refine) one (or more) specification(s) into a program.

The techniques of a formal method, (besides the specification languages) are typically software packages that help developers use the techniques and other tools.

The aim of developing software, either

- **formally** (all arguments are formal) or
- **rigorously** (some arguments are made and they are formal) or
- **systematically** (some arguments are made on a form that can be made formal)

is to (be able to) **reason in a precise manner about properties** of what is being developed.” (Bjørner and Havelund, 2014)
Decision Tables
## Decision Tables: Example

<table>
<thead>
<tr>
<th></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>( - )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( \times )</td>
<td>( - )</td>
<td>( \ast )</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>( - )</td>
<td>( \times )</td>
<td>( \ast )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( \times )</td>
<td>( - )</td>
<td>( - )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( - )</td>
<td>( \times )</td>
<td>( - )</td>
</tr>
</tbody>
</table>

**Table 1:**

The table above illustrates a decision table with conditions (\( c_1, c_2, c_3 \)) and actions (\( a_1, a_2 \)). The table consists of rules (\( r_1, r_2, r_3 \)) and their corresponding conditions and actions.

**Opt. Description:**

- \( c_1 \) \( \times \) \( c_2 \) \( \times \) \( a_1 \) -
- \( c_1 \) \( \times \) \( c_2 \) - \( a_2 \)
- \( c_2 \) \( - \) \( c_3 \) -
- \( c_3 \) \( \times \) \( a_1 \) -

**Legend:**

- \( \hat{\bullet} \) = rule name
- \( \hat{\circ} \) = premise
- \( \hat{\Box} \) = dont care
- \( \hat{\square} \) = effect
- \( \times \) = action
- \( - \) = no action
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

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

$r_1, \ldots, r_n$ are rule names.

$(v_{1,i}, \ldots, v_{m,i})$ is called premise of rule $r_i$,

$(w_{1,i}, \ldots, w_{k,i})$ is called effect of $r_i$. 
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>$v_{1,1}$</td>
<td>$\cdots$</td>
<td>$v_{1,n}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$c_m$</td>
<td>$v_{m,1}$</td>
<td>$\cdots$</td>
<td>$v_{m,n}$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$w_{1,1}$</td>
<td>$\cdots$</td>
<td>$w_{1,n}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$a_k$</td>
<td>$w_{k,1}$</td>
<td>$\cdots$</td>
<td>$w_{k,n}$</td>
</tr>
</tbody>
</table>

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

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

$$F(r) := F(v_1, a_1) \land \cdots \land F(v_m, c_m) \land F(w_1, a_1) \land \cdots \land F(w_k, a_k)$$

where

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

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

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

\[
\begin{array}{c|c|c|c}
\text{T} & r_1 & r_2 & r_3 \\
\hline
 c_1 & \times & \times & - \\
 c_2 & \times & - & * \\
 c_3 & - & \times & * \\
 a_1 & \times & - & - \\
 a_2 & - & \times & - \\
\end{array}
\]

- \( \mathcal{F}(r_1) = F(\times, c_1) \land F(\times, c_2) \land F(\neg, c_3) \land F(\times, a_1) \land F(\neg, a_2) \)
  \[
  = c_1 \land c_2 \land \neg c_3 \land a_1 \land \neg a_2
  \]

- \( \mathcal{F}(r_2) = F(\times, c_1) \land F(\neg, c_2) \land c_3 \land \neg a_1 \land a_2 \)

- \( \mathcal{F}(r_3) = F(\neg, c_1) \land F(\times, c_2) \land \neg c_3 \land a_1 \land \neg a_2 \)
Decision Tables as Requirements Specification
We can use decision tables to **model** (describe or prescribe) the behaviour of **software**!

**Example:**
Ventilation system of lecture hall 101-0-026.

<table>
<thead>
<tr>
<th>T: room ventilation</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

- We can observe whether **button is pressed** and whether room ventilation is **on or off**, and whether (we intend to) **start ventilation** or **stop ventilation**.

- We can model our observation by a boolean valuation $\sigma : C \cup A \rightarrow \mathbb{B}$, e.g., set
  \[ \sigma(b) := \text{true}, \text{ if button pressed now} \text{ and } \sigma(b) := \text{false}, \text{ if button not pressed now}. \]
  \[ \sigma(go) := \text{true}, \text{ we plan to start ventilation} \text{ and } \sigma(go) := \text{false}, \text{ we plan to stop ventilation}. \]

- A valuation $\sigma : C \cup A \rightarrow \mathbb{B}$ can be used to assign a **truth value** to a propositional formula $\varphi$ over $C \cup A$. As usual, we write $\sigma \models \varphi$ iff $\varphi$ evaluates to true under $\sigma$ (and $\sigma \not\models \varphi$ otherwise).

- Rule formulae $F(r)$ are propositional formulae over $C \cup A$ thus, given $\sigma$, we have either $\sigma \models F(r)$ or $\sigma \not\models F(r)$. 

\[ \sigma \models F(r) \]
We can use decision tables to **model** (describe or prescribe) the behaviour of **software**!

**Example:**
Ventilation system of lecture hall 101-0-026.

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>$\text{off}$ ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>$\text{on}$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>$\text{go}$ start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$\text{stop}$ stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

- We can **observe** whether **button is pressed** and whether room ventilation is **on or off**, and whether (we intend to) **start ventilation** of **stop ventilation**.
- We can model our observation by a boolean valuation $\sigma : C \cup A \rightarrow B$, e.g., set
  $\sigma(b) := true$, if button pressed now and $\sigma(b) := false$, if button not pressed now.
  $\sigma(go) := true$, we plan to start ventilation and $\sigma(go) := false$, we plan to stop ventilation.

- A valuation $\sigma : C \cup A \rightarrow B$ can be used to assign a **truth value** to a propositional formula $\varphi$ over $C \cup A$. As usual, we write $\sigma \models \varphi$ iff $\varphi$ evaluates to **true** under $\sigma$ (and $\sigma \not\models \varphi$ otherwise).
- Rule formulae $\mathcal{F}(r)$ are propositional formulae over $C \cup A$ thus, given $\sigma$, we have either $\sigma \models \mathcal{F}(r)$ or $\sigma \not\models \mathcal{F}(r)$.
- Let $\sigma$ be a model of an **observation** of $C$ and $A$. We say, $\sigma$ is **allowed** by **decision table** $T$ if and only if there **exists** a rule $r$ in $T$ such that $\sigma \models \mathcal{F}(r)$. 
Example

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>×</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>×</td>
<td>−</td>
<td>*</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>−</td>
<td>×</td>
<td>*</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>×</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>−</td>
<td>×</td>
<td>−</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
F(r_1) &= b \land \text{off} \land \neg \text{on} \land \text{go} \land \neg \text{stop} \\
F(r_2) &= b \land \neg \text{off} \land \text{on} \land \neg \text{go} \land \text{stop} \\
F(r_3) &= \neg b \land \text{true} \land \text{true} \land \neg \text{go} \land \neg \text{stop}
\end{align*}
\]

(i) **Assume**: button pressed, ventilation off, we (only) plan to start the ventilation.

- Corresponding valuation: $\sigma_1 = \{b \mapsto \text{true}, \text{off} \mapsto \text{true}, \text{on} \mapsto \text{false}, \text{start} \mapsto \text{true}, \text{stop} \mapsto \text{false}\}$.
- Is our intention (to start the ventilation now) allowed by $T$? **Yes!** (Because $\sigma_1 \models F(r_1)$)

(ii) **Assume**: button pressed, ventilation on, we (only) plan to stop the ventilation.

- Corresponding valuation: $\sigma_2 = \{b \mapsto \text{true}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{true}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true}\}$.
- Is our intention (to stop the ventilation now) allowed by $T$? **Yes.** (Because $\sigma_2 \models F(r_2)$)

(iii) **Assume**: button not pressed, ventilation on, we (only) plan to stop the ventilation.

- Corresponding valuation:
- Is our intention (to stop the ventilation now) allowed by $T$? $\not\models$
Decision Tables as Specification Language

- Decision Tables can be used to **objectively** describe desired software behaviour.

- **Example**: Dear developer, please provide a program such that
  - in each situation (button pressed, ventilation on/off),
  - whatever the software does (action start/stop)
  - is **allowed** by decision table $T$.

<table>
<thead>
<tr>
<th>$T$: room ventilation</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ button pressed?</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
<tr>
<td>$off$ ventilation off?</td>
<td>$\times$</td>
<td>$-$</td>
<td>$*$</td>
</tr>
<tr>
<td>$on$ ventilation on?</td>
<td>$-$</td>
<td>$\times$</td>
<td>$*$</td>
</tr>
<tr>
<td>$go$ start ventilation</td>
<td>$\times$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$stop$ stop ventilation</td>
<td>$-$</td>
<td>$\times$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Decision Tables as Specification Language

- Decision Tables can be used to **objectively** describe desired software behaviour.

- **Another Example**: Customer session at the bank:

  ![Decision Table Example](image)

  - clerk checks database state (yields $\sigma$ for $c_1, \ldots, c_3$),
  - database says: credit limit exceeded over 500 €, but payment history ok,
  - clerk cashes cheque but offers new conditions (according to $T1$).
**Decision Tables as Specification Language**

### 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,

- **Correctness and completeness** are defined relative to something which is usually only in the customer’s head.
  
  → is is **difficult** (if at all possible) to **be sure of correctness and completeness**.

### Another Example: Customer session at the bank:

- **T1**: cash a cheque
  - credit limit exceeded? $\times$$\times$
  - payment history ok? $\times$$\times$
  - overdraft $\times$$\times$

  → clerk checks database state (yields $\sigma$ for $c_1, \ldots, c_3$),

  - database says: credit limit exceeded over 500 €, but payment history ok,
  - clerk cashes cheque but offers new conditions (according to $T1$).

- **T2**: offer new conditions
  - $\sigma$ (as for $T1$)
  - $\sigma$ (as for $T1$)

- **T3**: software contract (incl. Pflichtenheft)
  - $\sigma$ (as for $T1$)

### Another Example: Client session at the bank:

- **T1**: cash a cheque
  - credit limit exceeded? $\times$$\times$
  - payment history ok? $\times$$\times$
  - overdraft $\times$$\times$

  → clerk checks database state (yields $\sigma$ for $c_1, \ldots, c_3$),

  - database says: credit limit exceeded over 500 €, but payment history ok,
  - clerk cashes cheque but offers new conditions (according to $T1$).
Decision Tables for Requirements Analysis
**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** (if at all possible) to **be sure of correctness** and **completeness**.
Definition. [Completeness] A decision table $T$ is called complete if and only if the disjunction of all rules' premises is a tautology, i.e. if

$$\models \bigvee_{r \in T} \mathcal{F}_{\text{pre}}(r).$$
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


