Topic Area Requirements Engineering: Content

- 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
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.

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)
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),
  - 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)*

---

**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 active voice. 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 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”.</td>
</tr>
<tr>
<td>R3</td>
<td>Discover incompletely defined verbs. In “the component raises an error”, ask whom the message is addressed to.</td>
</tr>
<tr>
<td>R4</td>
<td>Discover incomplete conditions. Conditions of the form “if-else” need descriptions of the if- and the then-case.</td>
</tr>
<tr>
<td>R5</td>
<td>Discover universal quantifiers. 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 nominalisations. 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 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?</td>
</tr>
<tr>
<td>R8</td>
<td>Clarify responsibilities. 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 implicit assumptions. 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 “⟨A⟩ ⟨B⟩ ⟨C⟩ ⟨D⟩ ⟨E⟩ ⟨F⟩.” (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) |

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

The definitions of these terms are an analog of definiti from a number of RFCs. In addition, suggestions have been incorporated from a number of people including Robert Ul Harten, 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

- 2019-05-16
**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*.

---

**Formal, Rigorous, or Systematic Development**

“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>T</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>c2</td>
<td>X</td>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>c3</td>
<td>-</td>
<td>X</td>
<td>*</td>
</tr>
<tr>
<td>a1</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>a2</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

- **Name**: Column headers for conditions or actions.
- **Rule**: Rules or conditions applied.
- **Effect**: Outcome or action following the rule application.
• 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>$C$ 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 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 $r_i$.

**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 $F(r)$ 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

$$F(r) := F(v_{1}, a_{1}) \land \cdots \land F(v_{m}, a_{1}) \land F(w_{1}, a_{1}) \land \cdots \land F(w_{k}, a_{k})$$

where

$$F(v, x) = \begin{cases} x, & \text{if } v = \times \\ \neg x, & \text{if } v = \times \\ \text{true}, & \text{if } v = \star \end{cases}$$
\[ F(r) := F(v_1, c_1) \land \cdots \land F(v_m, c_m) \land F(v_1, a_1) \land \cdots \land F(v_k, a_k) \]

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

\begin{tabular}{|c|c|c|c|}
\hline
\( r \) & \( r_1 \) & \( r_2 \) & \( r_3 \) \\
\hline
\( c_1 \) & \( \times \) & \( \times \) & \( \neg \) \\
\( c_2 \) & \( \times \) & \( - \) & \( * \) \\
\( c_3 \) & \( - \) & \( \times \) & \( * \) \\
\hline
\( a_1 \) & \( \times \) & \( - \) & \( - \) \\
\( a_2 \) & \( - \) & \( \times \) & \( - \) \\
\hline
\end{tabular}

- \( F(r_1) = F(c_1, c_1) \land F(c_1, c_2) \land F(-c_1) \land F(c_2, a_1) \land F(-a_2) \)
  \[ = \neg c_1 \land c_1 \land \neg c_2 \land c_1 \land a_1 \land \neg a_2 \]

- \( F(r_2) = c_1 \land \neg c_2 \land c_3 \land \neg a_1 \land a_2 \)

- \( F(r_3) = \neg c_1 \land \neg c_2 \land c_1 \land \neg a_1 \land \neg a_2 \)

\section*{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>室温调整按钮</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>按钮是否被按下？</td>
<td>是</td>
<td>否</td>
<td>是</td>
</tr>
<tr>
<td>通风是否关闭？</td>
<td>否</td>
<td>是</td>
<td>是</td>
</tr>
<tr>
<td>通风是否开启？</td>
<td>是</td>
<td>是</td>
<td>否</td>
</tr>
<tr>
<td>启动通风</td>
<td>是</td>
<td>否</td>
<td>是</td>
</tr>
<tr>
<td>停止通风</td>
<td>否</td>
<td>是</td>
<td>是</td>
</tr>
</tbody>
</table>

- 我们可以观察到按钮是否被按下，室温调整是否在开或关，以及我们是否打算启动通风或停止通风。
- 我们可以将我们的观察用一个布尔估值 $\sigma : C \cup A \rightarrow B$ 表示，例如，设定 $\sigma(b) := \text{true}$，如果按钮被按下并且 $\sigma(b) := \text{false}$，如果按钮没有被按下。

$$
\begin{align*}
\sigma(b) := \text{true}, \text{ if button pressed now and } \\
\sigma(go) := \text{true}, \text{ if we plan to start ventilation and } \sigma(go) := \text{false}, \text{ we plan to stop ventilation.}
\end{align*}
$$

- 一个估值 $\sigma : C \cup A \rightarrow B$ 可以被用来将一个**真值**分配给公式 $\varphi$ 诸如 $C \cup A$。
  要平凡，我们写 $\sigma \models \varphi$ 当且仅当 $\varphi$ 评估为真在 $\sigma$ 上（且 $\sigma \not\models \varphi$ 否则）。

- 规则命题 $F(r)$ 是约束命题的 propositional formulae over $C \cup A$。
  因此，给定 $\sigma$，我们有 $\sigma \models F(r)$ 或 $\sigma \not\models F(r)$。

- 让 $\sigma$ 是一个观察的模型 $C$ 和 $A$。
  我们说，$\sigma$ 是允许 **决定表** $T$，如果只有当有 $\sigma \models F(r)$。
Example

$T$: room ventilation

<table>
<thead>
<tr>
<th></th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>×</td>
<td>×</td>
<td>—</td>
</tr>
<tr>
<td>off</td>
<td>×</td>
<td>—</td>
<td>×</td>
</tr>
<tr>
<td>on</td>
<td>—</td>
<td>×</td>
<td>—</td>
</tr>
<tr>
<td>go</td>
<td>×</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>stop</td>
<td>—</td>
<td>×</td>
<td>—</td>
</tr>
</tbody>
</table>

$F(r_1) = b \land \neg\text{off} \land \neg\text{on} \land \text{go} \land \neg\text{stop}$

$F(r_2) = b \land \neg\text{off} \land \text{on} \land \neg\text{go} \land \text{stop}$

$F(r_3) = \neg b \land \text{true} \land \text{true} \land \neg\text{go} \land \neg\text{stop}$

(i) **Assume**: button pressed, ventilation off, we (only) plan to start the ventilation.
   - Corresponding valuation: $\sigma_1 = \{ b \mapsto \text{true}, \text{off} \mapsto \text{false}, \text{on} \mapsto \text{false}, \text{start} \mapsto \text{true}, \text{stop} \mapsto \text{false} \}$.
   - 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: $\sigma_3 = \{ b \mapsto \text{false}, \text{off} \mapsto \text{true}, \text{on} \mapsto \text{true}, \text{start} \mapsto \text{false}, \text{stop} \mapsto \text{true} \}$.
   - Is our intention (to stop the ventilation now) allowed by $T$? No

**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 $T$.
• Decision Tables can be used to objectively describe desired software behaviour.

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

<table>
<thead>
<tr>
<th>T1: cash a cheque</th>
<th>p1</th>
<th>p2</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1 credit limit exceeded?</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>c2 payment history ok?</td>
<td>x</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>c3 overdraft &lt; 500 €?</td>
<td>-</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>a1 cash cheque</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>a2 do not cash cheque</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>a3 offer new conditions</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(Balzert, 2009)

- clerk checks database state (yields σ for c1, ..., c3).
- 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.**

• **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.

→ is **difficult** (if at all possible) to be sure of correctness and completeness.
Recall Once Again

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 difficult (if at all possible) to be sure of correctness and completeness.
- **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.
**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

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

---

**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


