5 Documenting Requirements in Natural Language

Elicited requirements for the system to be developed are frequently documented using natural language. Natural language has the advantage that it (allegedly) does not require preparation time in order to be read and understood by stakeholders [Robertson and Robertson 2006]. In addition, natural language is universal in the sense that it can be used to describe any circumstances. However, there are some problems associated with the use of natural language for requirements documentation.

5.1 Effects of Natural Language

Subjective perception

As natural language is inherently ambiguous and statements in natural language can often be interpreted in multiple ways, it is necessary to place special emphasis on potential ambiguities in such statements to satisfy the criterion of unambiguousness. Requirements are defined and read by people with different knowledge, different social backgrounds, and different experiences. The diversity among the people involved in the development processes may lead to misunderstanding as humans interpret information differently (they form a so-called "deep structure" in their mind) and thus construe it differently as well (e.g., as a requirement). During such a process (i.e., perception and representation of information), so-called "transformational effects" occur that show different characteristics with every human but may occur in all humans [Bandler and Grinder 1975, Bandler 1994].

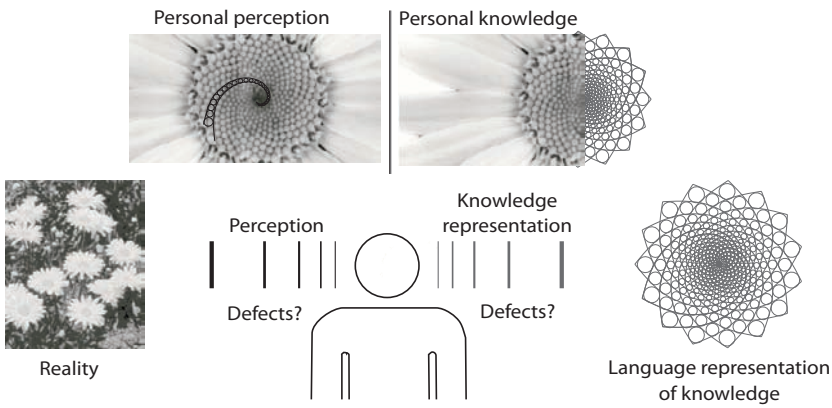

Figure 5-1 *Transformational effects in perception and representation of knowledge*

Transformational effects

The fact that transformational effects adhere to certain rules can be exploited by the requirements engineer to elicit the deep structure (i.e., what the author of a requirement really meant) from its surface structure (i.e., the requirements). The following list includes the five transformational processes that are most relevant for requirements engineering:

- Nominalization
- Nouns without reference index
- Universal quantifiers
- Incompletely specified conditions
- Incompletely specified process verbs

5.1.1 Nominalization

Reduction of processes

By means of nominalization, a (sometimes long-lasting) process is converted into a (singular) event. All information necessary to accurately describe the process is thereby lost. The process word *transmit* turns into the noun *transmission*. Other typical examples of nominalization are the terms *input*, *booking*, and *acceptance*.

Example 5-1: *Nominalization*

"In case of a system crash, a restart of the system shall be performed."
The terms *system crash* and *restart* each describe a process that ought to be analyzed more precisely.

Define processes completely.

Per se, there are no arguments against the use of nominalized terms to describe complex processes. However, the process should be explicitly defined by the term used. The definition of a nominalized term must not allow for any leeway in the interpretation of the processes and must precisely depict the process, including any exceptions that may occur as well as all input and output parameters. It is therefore not necessary to avoid nominalizations, but they should only be used if the underlying process is completely defined. During the linguistic analysis of a text, all nominalizations ought to be examined to determine whether they have been defined in sufficient detail in another part of the requirements document and whether they are clear for all stakeholders. If this is not the case, another requirement or a glossary entry must be created.

5.1.2 Nouns without Reference Index

Nouns with missing reference

As with process verbs, nouns are frequently incompletely specified. Linguists call this a missing or inadequate index of reference. Examples of terms that contain incompletely specified nouns are *the user*, *the controller*, *the system*, *the message*, *the data*, or *the function*.

Example 5-2: *Nouns without reference indices*

The data shall be displayed to the user on the terminal.

The following questions arise: What data exactly? Which user exactly? Which terminal exactly? If this information is amended, the requirement might thus read as follows:

Example 5-3: *Nouns with added reference indices*

The system shall display the billing data to the registered user on the terminal she is logged in to.

5.1.3 Universal Quantifiers

Specify amounts and frequencies.

Universal quantifiers specify amounts or frequencies. They group a set of objects and make a statement about the behavior of this set. When using universal quantifiers, there is the risk that the specified behavior or property does not apply to all objects within the specified set. Stakeholders tend to group objects together, even though some of these objects might

be special cases or exceptions, where the behavior specified does not apply to all the objects of a group.

Identify universal quantifiers.

It must be verified whether the specified behavior really applies to all objects summarized through the quantifiers. Universal quantifiers can be easily identified through trigger words such as *never*, *always*, *no*, *none*, *every*, *all*, *some*, or *nothing*.

Example 5-4: *Universal quantifiers*

The system shall show all data sets in every submenu.

In this case, the following question must be asked: Really in every submenu? Really all data sets?

5.1.4 Incompletely Specified Conditions

Identify and clarify condition structures.

Incompletely specified conditions are another indicator of a potential loss of information. Requirements that contain conditions specify the behavior that must occur when the condition is met. In addition, they must specify what behavior must occur if the condition is not met (the part that is often missing). Especially in complex conditional structures, decision tables can be invaluable tools to find unspecified variants of conditions or actions. Trigger words are, for instance, *if … then*, *in case*, *whether*, and *depending on*.

Example 5-5: *Incompletely specified condition*

The restaurant system shall offer all beverages to a registered guest over the age of 20 years.

At least one aspect remains unspecified in the example above: Which beverages shall be offered to a guest that is 20 years or younger? Clarifying this question may lead to extending the requirement as follows:

Example 5-6: *More completely specified condition*

The restaurant system shall offer

All alcohol-free beverages to any registered user younger than 21 years

All beverages including all alcoholic beverages to any user over the age of 20

5.1.5 Incompletely Specified Process Verbs

Completing process words

Some process verbs require more than one noun to be considered completely specified. The verb *transmit*, for instance, requires at least three supplements to be considered complete: *what* is being transmitted, *from where* it is being transmitted, and *to where* it is being transmitted. The feel for language (also referred to as "Sprachgefühl") is a valuable tool to help gauge which process word must be supplemented in order to be considered complete. Similarly, adjectives and adverbs may need to be supplemented as well. While the effect is much less frequent with these types of words than with verbs, it is often hard to recognize.

Avoid passive voice.

The use of incompletely specified process words can mostly be avoided or kept to a minimum if requirements are formulated using the active voice rather than the passive voice.

Example 5-7: *Requirement using the passive voice*

To log a user in, the login data is entered.

Use active voice.

In this requirement using passive voice, it is unclear who enters the login data. It is also unclear where and how this is done. If this requirement is reformulated using the active voice, at least the agent or person responsible must be included.

The same requirement using active voice might be as follows:

Example 5-8: *Requirement using active voice*

The system must allow the user to enter his user name and password using the keyboard of the terminal.

5.2 Requirement Construction using Templates

Quality by means of requirements templates and glossaries

Requirements templates provide a simple and easily understandable approach to reduce language effects when documenting requirements. Templates support the author in achieving high quality and syntactic unambiguousness in optimal time and at low costs.

Definition 5-1: *Requirements Template*

A requirements template is a blueprint for the syntactic structure of individual requirements.

In order to achieve lexical clearness in the documentation as well, it is wise to use requirements templates in conjunction with project glossaries (see section 4.7).

The following is a step-by-step description of the correct application of requirements templates.

Step 1: Determine the Legal Obligation

How legally binding is a requirement?

In the beginning, you should determine the degree of legal obligation for a requirement. Usually, one distinguishes between legally obligatory requirements, urgently recommended requirements, future requirements, and desirable requirements. To achieve this within a requirement, you can use the modal verbs *shall*, *should*, *will*, and *may*. Alternatively, the legal obligation of a requirement can be documented by a specific requirements attribute.

Step 2: The Requirement Core

Determine the required process.

The core of each requirement is the functionality that it specifies (e.g., print, save, paste, or calculate). This functionality is referred to as the *process*. Processes are activities and may only be described using verbs. The process that depicts the system behavior by means of a requirement is to be described in step 2.

Since process words determine semantics, they must be defined as clearly as possible and be used as consistently as possible (see section 4.7).

Step 3: Characterize the Activity of a System

For functional requirements, the system activity can be classified as one of three relevant types:

- *Autonomous system activity:* The system performs the process autonomously.
- *User interaction:* The system provides the process as a service for the user.
- *Interface requirement:* The system performs a process depending on a third party (e.g., another system). The system is passive and waits for an external event.

In step 3, any kind of system activity that is specified by a requirement of the system is documented using exactly one of three requirements templates. These requirements templates are described in more detail in the following sections.

After performing steps 1 through 3, the structure of the requirement has been developed (see figure 5-2). The words that are written in angle brackets must be replaced accordingly.

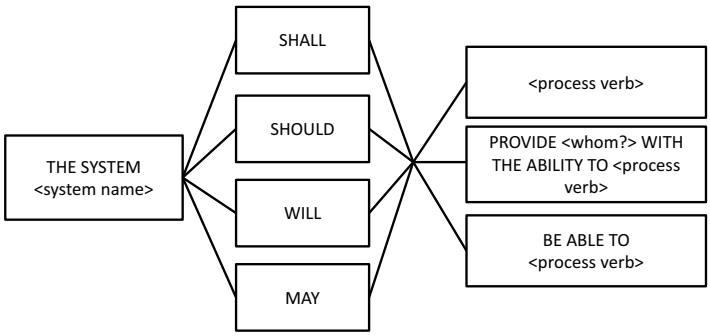

Figure 5-2 *The core of a requirement and its legal obligation*

Type 1: Autonomous system activity

The first template type is used when requirements are constructed that depict system activities that are performed autonomously. The user does not interact with the activity. We define the following requirements template:

THE SYSTEM SHALL/SHOULD/WILL/MAY <process verb>

<Process verb> depicts a process verb as described in step 2, e.g., *print* for print functionality or *calculate* for some calculation that is performed by the system.

Type 2: User interaction

If the system provides a functionality to a user (for example, by means of an input interface), or the system directly interacts with a user, requirements are constructed using template type 2:

THE SYSTEM SHALL/SHOULD/WILL/MAY provide <whom?> with the ability to <process verb>

The user that interacts with the system is integrated into the requirement through <whom?>.

Type 3: Interface requirement

If the system performs an activity and is dependent on neighboring systems, the third template type is to be used. Whenever messages or data

are received from a neighboring system, the system must react by executing specific behavior. The following template has proven itself as well suited:

THE SYSTEM SHALL/SHOULD/WILL/MAY be able to <process verb>

Step 4: Insert Objects

Complete process verbs.

Some process verbs require one or more additional objects to be considered complete (see section 5.1.5). In step 4, potentially missing objects and supplements of objects (adverbials) are identified and added to the requirement. For instance, the requirements template for the process verb *print* is amended by the information of *what* is being printed and *where* it is printed. The amendment can be seen in figure 5-3.

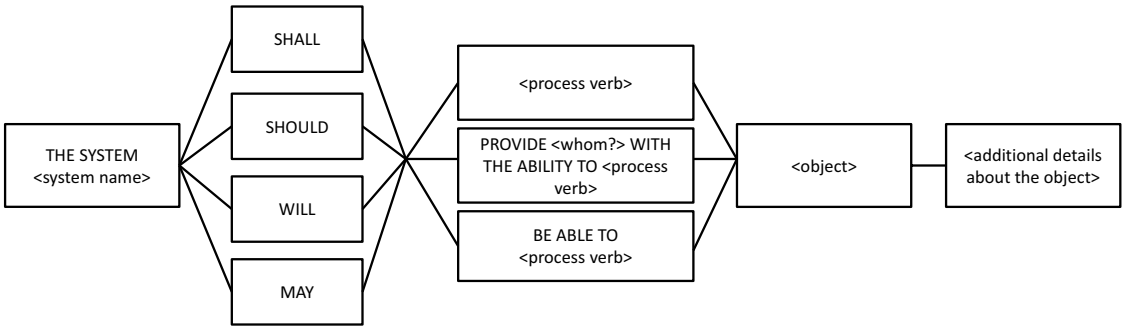

Figure 5-3 *Principle of a complete requirements template without conditions*

Step 5: Determine Logical and Temporal Conditions

Add conditions.

Typically, requirements do not document continuous functionalities, but functionalities that are performed or provided only under certain logical or temporal constraints. In order to easily differentiate between logical and temporal conditions, we choose the temporal conjunction *as soon as* for temporal conditions and the conditional conjunction *if* for logical conditions. The conjunction *when* makes not clear whether a temporal or a logical condition is described and should therefore be avoided. In step 5, quality requirements that describe the conditions under which a requirement is fulfilled are added to the beginning of a requirement as a subordinate clause.

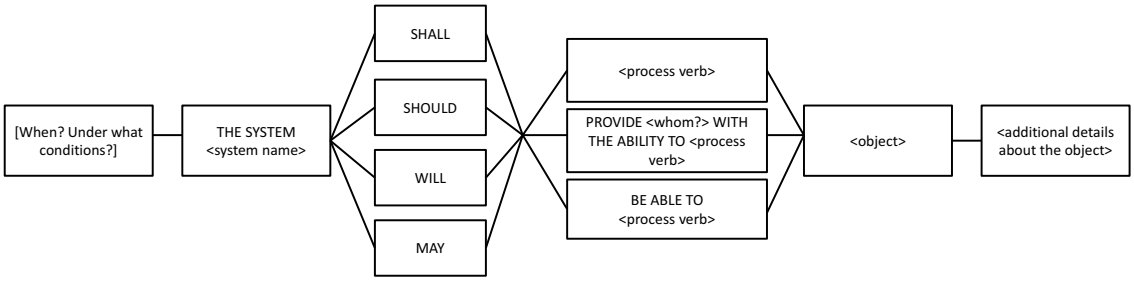

Figure 5-4 *The complete requirements template with conditions*

Requirements templates should be used when project members show interest in a formal development process. Style and creativity are harshly limited when requirements templates are used. Experience shows it is best not to make the use of requirements templates compulsory, but to offer training on the method and treat it as a supplemental tool.

5.3 Summary

System requirements are frequently documented using natural language. Typical advantages that arise from natural language requirements are good readability of requirements, the fact that natural language can be universally applied to document any circumstance, and the fact that no prior knowledge is necessary regarding the notation. On the other hand, there are a number of disadvantages that arise from the fact that natural language requirements are not formalized, e.g., ambiguity. Since project members interpret requirements differently due to differences in their respective knowledge, social background, and experiences, using natural language for requirements documentation often leads to misunderstanding in practice. These disadvantages can be minimized during requirements documentation—for example, by making use of requirements templates and by checking the requirements against linguistic effects.