I. Preamble

1. Intend

The practice describes the pattern for specifying requirements.

2. Motivation/Pros

- Reduction of language effects (misunderstandings)
- Supports unambiguousness of syntax
- Supports creation of high quality requirements
- Supports time-efficient creation of requirements

3. Cons

- --

4. Applicability

Requirements pattern shall be applied for any kind of requirements specified in natural language.

II. Description of the Practice

1. Elements of a Requirement

Element	Description
<event/condition>	The event that shall trigger the <action> when it occurs. OR: The condition that shall be fulfilled to conduct the <action>
<actor>	The acting part, i.e. the one who is obliged to evaluate the <event/condition> and conduct the <action>
<legal binding>	Key word signifying the relevance of the requirement. Following key words are applicable (in accordance to RFC 2119): "**shall**" This word mean that the item is an absolute requirement. "**shall not**" This phrase mean that the item is an absolute prohibition. "**should**" This word mean that there may exist valid reasons to ignore the item, but the full implications must be understood and carefully weighed before choosing a different course. "**should not**" This phrase mean that there may exist valid reasons when the item is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label. "**may**" This word mean that the item is truly optional. "**will**", **"is**" These keywords identify a statement of fact, not a requirement. For objects of type "Req-XXX" only keywords "**shall**" and "**shall not**" shall be used. For objects of type other than "Req-XXX", like "Heading" or "Information" only keywords "**should**", "**should not**", "**may**", "**will"**, and **"is**" shall be used.
<action>	The actions being conducted by <actor> when the <event> occurs or the <condition> is fulfilled.
<object of action>	Optional specification of the object that undergoes the <action>.

Table 1: Elements of a Requirement

2. Requirements Pattern

To make a requirement clearer, the necessary parts of a requirement shall be organized in specific patterns.

2.1. Requirements Pattern 1

```
<actor> <legal binding> <action> [<object of action>] <event/condition>
```

Figure 1: Requirement Pattern 1

Examples:

The system shall turn on when the power button is pressed while the system is off.

The system shall switch off the lights, if the battery signal value is below 20%.

The system shall stay off while the battery signal value is below 20%.

2.2. Requirements Pattern 2

```
<event/condition> <actor> <legal binding> <action> [<object of action>]
```

Figure 2: Requirement Pattern 2

Examples:

When the power button is pressed while the system is off, the system shall turn on.

If the battery signal value is below 20%, then the system shall switch off the lights.

While the battery signal value is below 20%, the system shall stay off.

3. Active/ Passive Voice

In general, active voice is considered as the standard for requirements (see IREB recommendations).

When using passive, there is a risk that the requirements do not specify "who" is responsible.

We therefore recommend that the authors use the active voice in the requirements specifications.

Nevertheless, the requirements engineers can decide for each individual requirement which kind of formulation to use.

Arguments for Active Voice:

- Clearly states “who” shall do “what”.
- Active Voice helps to choose the correct ObjectType. For example: Currently, production requirements are often misclassified as “Req-Product”.
- Active Voice will support the fact that the requirement engineers think more about the requirement. Which hopefully increases the quality.
  This can really be an advantage especially for the new colleagues.

Suitable actors are:

- The system
- The software
- The project/ The company (in case of Req-Quality)

4. Weak Words and Phrases

Certain words from natural language allow misinterpretation and do not add information to the requirement. Therefore, it is desired to avoid those.

Below is an incomprehensive list of such words:

Examples of Weak Words	Comment
efficient, powerful, effective, easy, reliable, adequate, special, simple	seemingly strong words without adding meaning to the requirement
good, bad, extreme, big, small	relative wording without clear meaning to the requirement
always, never, in all cases, at all times	these cannot be verified
actually, very, further, usually	filler words which do not add meaning to the requirement
compatible	compatible to what?
user-friendly	depends on every single user
few, most, many, quickly, timely, fast, slow, promptly, often	unclear amount and time
it, its, they, them, their, this, he, she	avoid Pronouns, these bear the risk of misinterpretion

Table 2: Weak Words

Some words are weak if their specific context is unclear:

Examples of context specific words	Comment
normal	A *Normal* state is to be defined before, otherwise a requirement containing this word becomes unclear
more, less,	*More/Less* than what exactly? This part of a condition has to be added to the requirement
maximal, minimal	It has to be defined before what the *Maximum/Minimum* is.

Table 3: Context specific words

5. Negative Requirements

Sometimes it seems necessary for the author to formulate a negative requirement (by using “shall not”). However, it is not possible to specify all the things that the system should not do.

The goal is to specify what the system shall do. When using negative requirements, the author may forget to specify parts of them.

Phrasing requirements using "shall not" may cause reviewers to question other things that the system shall not do. Also the implementation of these requirements is complicated to proof. Such confusion can generally be avoided by following these rules:

- Use negative specifications primarily for emphasis to prohibit potentially dangerous actions. In this case, add a note with the justification for the requirement to the “Object Text”.
- Do not use a negative specification for requirements that can also be stated positively.
  Bad: “There shall be no sliding possible if more than one finger is detected on the slidearea”
  Good: "If the system detects more than one finger on the slidearea, the system shall set the signal XY to 0."
- Avoid double negatives completely.
  Bad: “The system shall not switch on the illumination, in case the button is not pressed.”
  Good: “The system shall switch on the illumination if the button is pressed.”

6. Avoid Design Constraints

6.1. Goal

The requirements should not restrict the architects to a particular solution/implementation; they should be free of design details. Requirements should specify “what the system shall do”, not “how the system shall do it”.

➔ Solution/Implementation-Neutral

6.2. Rational

A “Solution-Neutral” specification allows the architects to design the system in the most efficient manner available.

When the requirements specify the constraints, the tester is not able to verify if problem is solved.

➔ The product may satisfy the constraints, but not the customer.

Example:
Customer: “Button shall be debounced between 30-50ms.”
If the implemented button requires a debouncing of e.g. 80ms (50ms are used), the system may fulfill the customer requirements but the signal will flicker.
By only using the customer constraint without knowing the intention of the customer (e.g. preventing flickering), the developed system will not satisfy the customer.

6.3. Action

When the customer specifies a Design Constraint, the Sys-RE should try to resolve it together with the customer. Often the customer has a very specific use case in mind when specifying a Design Constraint. Maybe this reason does not fit to the project.
The Sys-RE should try to find out and specify the problem to be solved.

When the customer still insists on a Design Constraint, it should be marked in very specific non-functional requirements.

Appendix

A. History of Template

Date	Vers.	Status	Description of Change	Author
2018-xx	0.1	Draft	Initial version of the template	Schmidt_m2
2019-02-15	0.2	Draft	Update of the history of the template and cleared table with history of the practice	Schmidt_m2
2019-05-10	0.3	Draft	Style “Cross Reference” added. Size of text in column “Author” decreased.	Schmidt_m2
2019-05-13	0.4	Draft	Combined template for Preh and innoventis	Schmidt_m2

Table 4: History of Template

B. History of Practice

Date	Vers.	Status	Description of Change	Author
2019-09-27	0.1	Draft	Initial version	Schmidt_m2
2019-09-27	0.2	Draft	Added Weak Words	Lengwenus_m1
2020-01-31	0.3	Draft	PDB number corrected	Schmidt_m2
2021-12-01	0.4	RfR	reworked	RuedenauerB
2022-01-11	1.0	Released	Released after review of Markus Schmidt	RuedenauerB

Table 5: History of Practice

C. Referenced Documents

Unless otherwise specified, the latest approved version of the referenced document is valid.

Referenz-ID	Name of the document, if necessary incl. version	Author

Table 6: Referenced Documents

D. Glossary

Abbreviation / Term	Explanation

Table 7: Glossary

E. List of Tables

F. List of Figures