Chris Rupp

Requirements Templates — The Blueprint of your Requirement

STANDARD REQUIREMENTS

The fully automated creation of a requirements document seems to remain a dream in the near future. Nevertheless, using surprisingly simple tools it is possible to create requirements of high quality, cost and time efficient. This article explains the template-based way for the construction and quality assurance of explicit, complete and testable requirements, which then again may be embedded into a system of analysis patterns.

Templates For A Optimal Solution

For a long time, plans, templates and masks have been the typical aids of architects, mechanical engineers or chip designers. In software development it is being spoken of software architecture and the meaning of design patterns is commonly known since the "Gang of Four" published their book. Unfortunately, this approach is only being used in the phases of design and implementation, with the attempt to create templates from it.

In the recent past, the advantages of a process based on patterns have been recognized in the field of requirements engineering. Now the attempt is being made to present the issue to be solved in a more effective way by using already know patterns. Natural language requirements are not written any longer, they are being constructed based on templates in order to set up a complete, uniform set of requirements. This enables a common understanding between the author and the reader about the intention of the requirements. These requirements may be implemented systematically into an object oriented analysis model or used in the acceptance criteria1 for tests and system verifications. The process of starting with an idea up to creating a perfectly written requirement and its further processing, for example in a model, is being systemized.

How To Create A Perfect Requirement

There are various ways to obtain requirements of excellent content. When proceeding analytically [Rupoo], requirements of varying quality are being formulated, analysed and improved step by step, based on the natural language method. The analysis and improvement would not be necessary, or at least not to that extent, if requirements of high quality, meaning correct and comprehensible, were available from the start. Now there are new methods available, which show what an ideal requirement should look like. Using these eliminates many of the possible mistakes being made in wording the requirements, like passive sentences, which usually don't offer any information about what functionality is being expected from whom. The author of a requirement is taken by the hand and led to formulate a high quality requirement using few, but very clear guidelines.

This procedure is much more efficient than the analytical method. Constructing requirements according to certain rules is a means of avoiding mistakes from the very beginning. This way of creating requirements makes it possible to assign a similar structure to each requirement. Similar to building a house according to an architect's plan, it is possible to construct a requirement according to a plan, or rather according to a template. These are called generic, **syntactical requirements templates**, since only the syntax (structure) of a requirement is defined, not its semantics (the contents). Practically speaking, only

three different templates are necessary to completely specify a system using natural language requirements. Even though it is amazing to be able to represent the variety of the natural language using only these templates, this method has successfully been used in our projects. Exactly that has been the goal, to be able to unite differently structured statements and expressions, which is being accepted and welcomed by most stakeholders.

> Definition: A generic, syntactical requirements template is the blueprint that determines the syntactical structure of a single requirement.

Requirements Step By Step

The following instructions show how a requirement based on a template is constructed using only five steps.

Step 1: Start with the process

The central point of every requirement is the functionality to be provided by the system (for instance to print, to save, to transmit, to calculate), which we will call the *process* in our discussion. Processes are procedures and action, which should be exclusively defined using verbs. A verb plays a very important role, because the whole sentence, or in our case the requirement, is connected to this process word. Therefore it is essential to determine the required process in the beginning.

Step 2: Characterize the activity of the system

The way in which the system works is closely linked to the process word. Efforts to classify these show, that the following three types of system activities are relevant:

- The system *carries* out the process by *itself*
- The system *provides* the user with process funtionality
- The system carries out the process *dependent on a third factor* (for instance a different system), remains passive and waits for an external event

Accordingly, it is now possible to choose one of three requirements templates. Please have a look at figure 1, which shows the basic structure of a requirement.

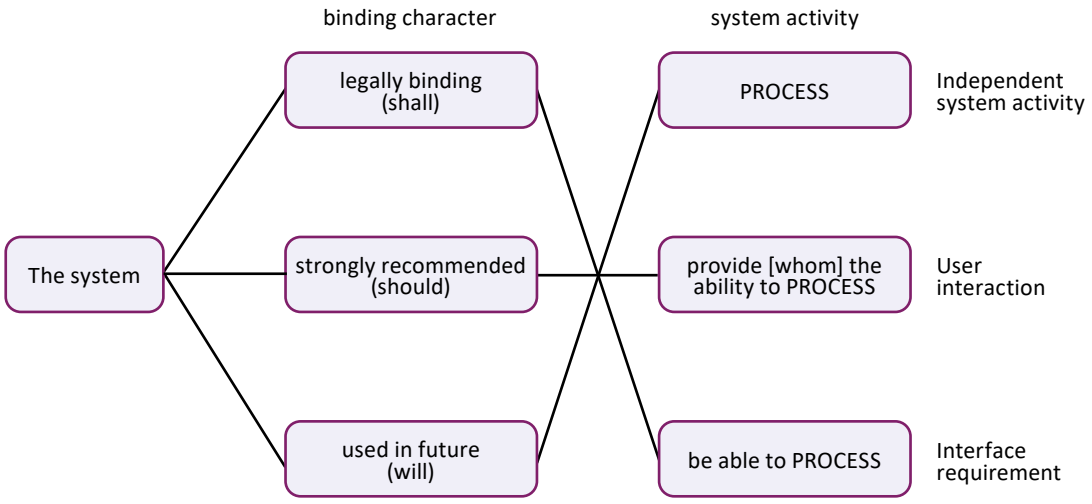


Figure 1: The core of a requirement

Type 1: Independent system action

The first template type is used to construct requirements describing a functionality a system carries out independently. The user does not take action in this case. Not taking the legal responsibilities into consideration, which will be discussed in step 3, the following requirement skeleton is obtained.

> THE SYSTEM ... <process>.

<process> represents the process word chosen in step 1, for example "to print" for a print function, or "to calculate" for a calculation the system is to carry out.

Type 2: User interaction

If the system provides the user with a functionality (i.e. a user interface) or if it interacts with the user, the requirements are constructed according to template type 2.

> THE SYSTEM ... PROVIDES [whom] THE ABILITY TO <process>.

The user (e.g. administrator) who is to use the functionality is integrated into the requirement with [whom]. This is not the case if the user has been defined outside the scope of the requirement, hence known.

Type 3: Interface requirement

Type 3 is relevant in case the system carries out functionality, but is dependent on a third factor in order to do so. Imagine a system, which does not receive information from the user but from another system. The input could be asynchronous and unpredictable. Every time a message or data is received, our system reacts and carries out an action. To create this requirement, type 1 (independent system action) as well as type 2 (user interaction) is inappropriate. Type 1 cannot be used since the system does not act independently, type 2 due to the missing user activity. The actual function is carried out by the other system by sending data to the interface of the receiving system. Based on the above, the following statement is suitable.

> THE SYSTEM ... IS (BE) ABLE TO <process>

Step 3: Determining the legal obligation

To complete the core of the requirement it is also necessary to determine its legal relevance. Usually it is being distinguished between requirements, which are legally binding, strongly recommended or used in the future. One way to express the legal relevancy within a requirement is to use the auxiliary verbs shall, should and will [SOP04].

The three steps introduced so far may be exemplified as follows. Let us assume our planned system should offer a print option. We determine *"print"* as the process word (step 1). Now the question arises, whether the system should print independently or give the user the option to. We assume the latter to be true and have implicitly chosen requirement template 2 (user Interaction). This requires the determination of a user, in our case the administrator. Since the option to print is indispensable to us and must be integrated into the system, we declare this requirement as legally binding (step 3). In figure 1 this example is highlighted.

Using only three steps we hereby obtained the following basic structure of a requirement:

Requirement no.1, version 1: The system **shall** provide **the administrator** the ability to **print**.

Step 4: Precisely determining the process

The example shows, that so far we are only dealing with the core of the requirement. Further elements are required for the completion. In requirement no.1, for instance, it is not apparent at all, *what* is to be printed and *where* it is to be printed *to*. Linguists would use the terms „missing objects" and „supplementations". Putting it simply, a closer or supplementing characterization of the process word „print" is missing.

Let us extend our example:

Requirement no.1, version 2: The system **shall** provide the **administrator** the ability to **print** the error message to the network printer.

In English sentences, objects and supplementations are always appended to the verb and therefore are located behind the core of a requirement (see figure 2 below).

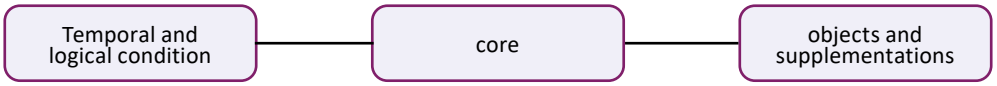


Figure 2: Structure of a complete requirements template

Step 5: Formulating logical and temporary conditions

It is typical for requirements constructed in technical systems, that the functionality is only given or provided under certain logical or temporal conditions. These constraints are located at the beginning of a requirement.

Requirement no.1, version 3: **If an error message has been generated**, the system shall provide the administrator the ability to print the error message to the network printer.

Optimizations — Tuning The Constructed Requirement

Looking at existing requirements documents, we can tell that we progressed in terms of the quality, yet we're only half way to a good requirement. The requirement constructed in step 5 certainly meets medium to high quality standards, yet various optimisations can be achieved. A few will be dealt with in the following.

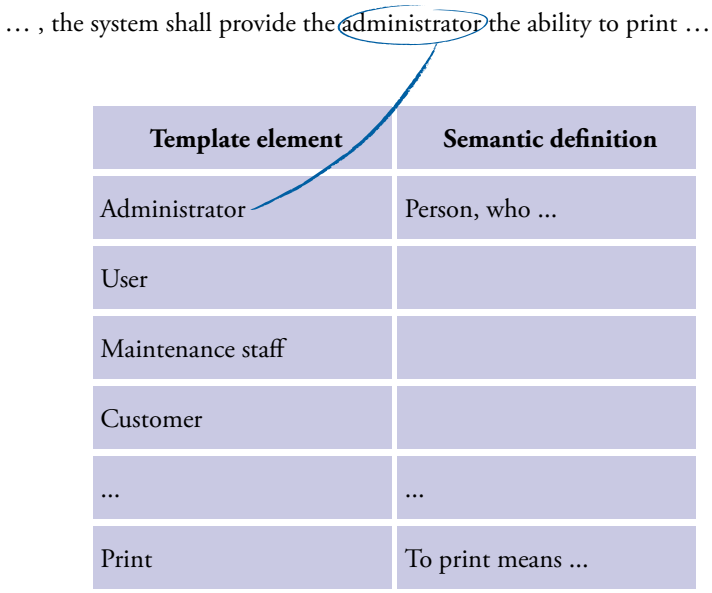
Can't do without analysis...

Templates have several advantages, but they also can force a structure upon a sentence (here a requirement), which is not desired in that way. Or they offer too many options to the analyst in certain respects, that mistakes can readily be made. Accordingly, the measures taken in step 4 enable the writer to select the sentence object and its supplementations almost unrestrictedly. Consistency and completeness are the requirement writer's responsibility. It is reasonable, therefore, to subject each constructed requirement to an analysis, ideally a natural language method [Rupp00]. That way, the requirement is improved step by step without losing its structure.

How about the content?

We can't deny that the requirement template is nothing more than a simple frame that gives structure and clarity to a requirement – nothing more but nothing less. Content-wise (semantically) this requirement is as good or bad as one created in the conventional way. To counteract this problem, it is necessary to define every single template element (process word, user, objects, ...) semantically. Therefore, glossaries should be maintained, in which the possible elements are defined unambiguously. To do this in a practical efficient way and to ensure traceability, the use of specialized software tools seems indispensable. The tool to be used must at least be able to produce and maintain the relationship between the template based requirement and its semantic definition.

Figure 3 shows an example of a semantic definition of a requirement element. The most important elements to be defined are the process words, which, as we have discussed, play a key role in the requirement, since they are connected to the expected functionality. For complete understanding of the requirement, it is necessary to know, what the author is asking for. Or do you know the difference between entering, inputting and inserting? One person may see no difference at all; another may just use the word as he interprets it at that point. Yet another may differentiate according to user or system input (analog to our types of templates). The last interpretation seems to be the best way, which we have used successfully in many projects. The information expressed by the process word is clear, so that there is no need for long explanations. The requirements document becomes slim, given that the definitions are known.



Template element	Semantic definition
Administrator	Person, who ...
User	
Maintenance staff	
Customer	
...	...
Print	To print means ...

Figure 3: Semantic definition of the element [whom]

The structure within the structure

Complex systems tend to provide certain functionalities only under certain preconditions. During step 5 of the construction, the analyst is faced with a lot of temporary and logical conditions, which need to be integrated into the requirement without contradictions. It has to be *absolutely clear*, under which conditions the required functionality is provided by the system. To achieve this, the conditions have to be structured using the Boolean operators AND, OR and XOR combined with NOT:

Requirement no.1, version 4: If an error message has been generated **AND** the automatic print function is disabled, the system shall provide the administrator the ability to print the error message to the network printer.

Analysis Pattern

The consequence of the template approach

During the development of the system, following the analysis, natural language requirements are usually semi formalized into object-oriented analysis models, a process, which until now had more to do with creative ability than with a systematic approach. Moreover, some time the acceptance of the system and thus the acceptance of the requirements will take place. This shows the advantages of constructing a requirement using templates.

The uniform structure of the requirement provides an easy identification of the process, the conditions or the objects. On this basis, object models can be built without having to sort out fragmentary and confusing requirements. Even the writers of acceptance criteria1 are glad to be able to view all conditions within a requirement at once, since it is the base for the creation of equivalent groups. With a growing number of requirements, these advantages add up to a significant acceleration of the requirements analysis process, since it is clear, where which element of the requirement can be found. The uniform construction makes it possible to show derivation and illustration rules for a big number of requirements using models and acceptance criteria, enabling these tasks to be learned and delegated.

Requirements patterns — the big picture

Figure 4 illustrates, how the components described in this paper interact with each other. The set of all these components is what we call the „Requirements Pattern".

> Definition: A requirements pattern represents an approach for the construction of natural language requirements, which can be modelled and tested based on their formally defined components.

A requirements pattern describes application specific problems. Natural language requirements constructed using templates and their semantically defined elements are being provided with the associated acceptance criteria and model elements. Requirements patterns are part of the analysis, i.e. the *problem space* - in contrast to design patterns, which are part of the *solution space*.

Every pattern characterizes exactly one problem, e.g. user input into a graphic front end or an interface connection of two systems. These patterns can be combined in pattern-catalogs from which the analyst may choose when specifying a system. Requirements patterns are especially useful for specialized issues as well as for recurring non-functional requirements.

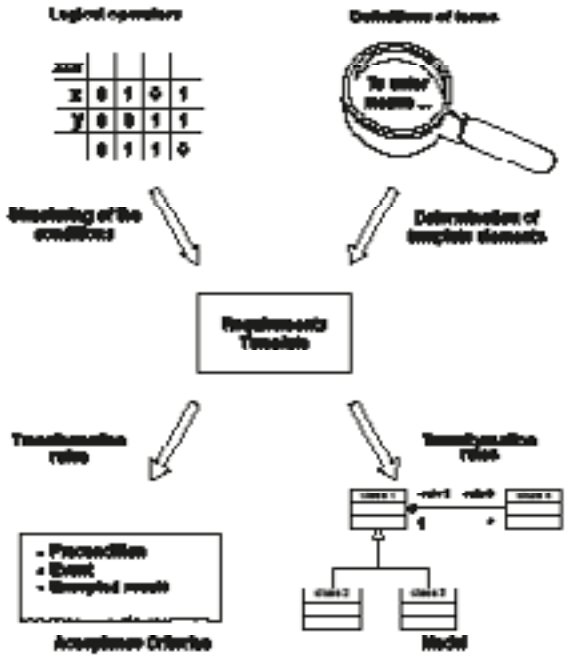


Figure 4: Structure of requirements patterns

Conclusion

It is that simple. In a few steps a perfect requirement is created. As if by magic, requirement documents and analysis models are generated. What's the further need for system analysts? Customers are writing their own requirements, quality assurance managers are enthusiastic and the project management is viewing the first on time completed system analysis.

Though we have only been able to show a small portion of this approach, it should be obvious that the need for analytical thinking has not been eliminated. The advantages of this approach are shown in the improvements in time and costs and the high quality standards throughout the resulting requirements. The storing of information in a uniform and familiar structure combined with methodically implied, explicit and testable requirements make requirements patterns universal and easy to understand tools

for system analysis. They represent, especially in the natural language environment, a link between the chaotic form of the natural language and the defined and structured world of analysis models. This bi-directional understanding is what responds to the customer. The writing of requirements is especially convincing. Imagine what's easier for you: to fill out a template under given conditions - at best software supported – or to write an article sentence by sentence (possibly not in your first language)?

On the other hand are the disadvantages: The reading of uniform and similar structures is tedious. It is being said, that the reading of these specifications is an aid for readers with sleeping problems. Because of this, reviews should take place using an object-oriented model or a prototype, instead of just reading the requirements document.

An additional risk pose the semantic definitions, which could cross one another, if they are only used by a part of the project group. This is preventable through good management and organization. All in all, the advantages of requirements templates and patterns are enormous. The time saved with the use of this approach covers the few disadvantages described.

References

You can find additional information on this subject at www.sophist.de and in [SOP04]. You can also find a formal description of the templates in Backus- Naur-Form or definitions of standard process words.

[Fow97] **M. Fowler**, Analysis Patterns – Reusable Object Models, Addison Wesley 1997.

[Gam94] **E. Gamma**; R. Helm; R. Johnson; J. Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley Longman 1994.

[Hru00] **P. Hruschka**: Von informellen Wünschen und formalisierten Anforderungen. In: OB JEKTspektrum 02/ 2000

[Rup00] **C. Rupp:** Requirements - Engineering – der Einsatz einer natürlichsprachlichen Metho de bei der Ermittlung und Qualitätsprüfung von Anforderungen. In: OBJEKTspek-trum02/ 2000

[SOP04] **C. Rupp; SOPHIST GROUP:** Requirements - Engineering und - Management – Professionelle, iterative Anforderungsanalyse für die Praxis. München, Carl Hanser Verlag 2004

Copyright © 2019 by SOPHIST GmbH

Publikation urheberrechtlich geschützt. Alle Rechte, auch die der Übersetzung, des Nachdruckens und der Vervielfältigung oder Teilen daraus, vorbehalten. Kein Teil der Publikation darf in irgendeiner Form, egal welches Verfahren, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet werden, vervielfältigt oder verbreitet werden.

Dies gilt auch für Zwecke der Unterrichtsgestaltung. Eine schriftliche Genehmigung ist einzuholen. Die Rechte Dritter bleiben unberührt.