Journal of Software Engineering Research and Development, 2019, 8:3, doi: 10.5753/jserd.2020.473
This work is licensed under a Creative Commons Attribution 4.0 International License.

Towards a new template for the specification of requirements in semi-structured natural language

Raúl Mazo [Lab-STICC, ENSTA Bretagne, Brest, Francia. GiDITIC, Universidad EAFIT, Medellín, Colombia | raul.mazo@ensta-bretagne.fr]
Carlos Andrés Jaramillo [Universidad EAFIT, Medellín, Colombia | cajaramilg@eafit.edu.co]
Paola Vallejo [GiDITIC, Universidad EAFIT, Medellín, Colombia | pvallej3@eafit.edu.co]
Jhon Harvey Medina [Universidad EAFIT, Medellín, Colombia | jhmedinaa@eafit.edu.co]

Abstract

Requirements engineering is a systematic and disciplined approach for the specification and management of software requirements; one of its objectives is to transform the requirements of the stakeholders into formal specifications in order to analyze and implement a system. These requirements are usually expressed and articulated in natural language, this due to the universality and facility that natural language presents for communicating them. To facilitate the transformation processes and to improve the quality of the resulting requirements, several authors have proposed templates for writing requirements in structured natural language. However, these templates do not allow writing certain functional requirements, non-functional requirements and constraints, and they do not adapt correctly to certain types of systems such as self-adaptive, product line-based, and embedded systems. This paper (i) presents evidence of the weaknesses of the template recommended by the IREB® (International Requirements Engineering Institute), and (ii) lays the foundations, through certain improvements to the template proposed by the IREB®, for facilitating the work of the requirements engineers and therefore improving the quality of the products specified with the new template. This new template was built and evaluated through two active research cycles. In each cycle we identified the problems specifying the requirements of the corresponding industrial case with the corresponding base-line template, propose some improvements to address these problems and analyze the results of using the new template to specify the requirements of each case. Thus, the resulting template was able to correctly write all requirements of both industrial cases. Despite the promising results of this new template, it is still preliminary work regarding its coverage and the quality level of the requirements that can be written with it.

Keywords: *Requirement, requirements engineering, natural language, template, application requirement, domain requirement, self-adaptive requirement*

1 Introduction

The requirements are perhaps the most important basis in the construction of software products because, through them, the stakeholders of the system that is going to be implemented can achieve a common understanding of it. According to Wiegers and Beatty (Wiegers and Beatty 2013), the two most important objectives in specifying a requirement are that (i) when several people read the requirement they reach the same interpretation; and (ii) the interpretation of each reader coincides with what the author of the requirement was trying to communicate.

In this sense, Pohl (Pohl 2010) states that NL (Natural Language) is the most common way to communicate and document the requirements of a system since NL is universal and available to any individual in any field; besides, it does not require any kind of special training in the interpretation of notations or symbols as occurs when using an engineering language such as UML (Unified Modeling Language). However, these advantages are overshadowed by the disadvantages of natural language (Rupp 2007). According to Mavin *et al.* (Mavin et al. 2009) some of the problems susceptible to appear in the requirements specification in NL are (i) *ambiguity:* a word or phrase has two or more different meanings; (ii) *vagueness:* lack of precision, structure or detail; (iii) *complexity:* composite requirements that contain complex sub-clauses or several interrelated statements; (iv) *omission:* missing requirements, particularly the requirements to handle unwanted behavior; (v) *duplication:* repetition of requirements defining the same need; (vi) *verbosity:* use of an unnecessary number of words; (vii) *implementation:* statements of how the system should be built, rather than what the system should do; and (viii) *untestability:* requirements that cannot be proven (true or false) when the system is implemented.

To reduce these problems in the specifications of the requirements of a system, several authors have defined what is known as template, mold, pattern or boilerplate (Rupp 2007). A template defines the structure that the requirements written in NL should have; that structure is flexible so that the resulting requirements have the advantage of being in NL and the advantage of having a well-defined structure. This NL bounded by the possibilities and restrictions of the template is known as semi-structured natural language.

The notations in semi-structured language make it possible to build requirements by following a template and assigning a similar structure to each requirement. This approach helps to avoid errors in the early stages of the development process by specifying high-quality requirements efficiently in time and cost (Sophist 2014).

The template proposed by Rupp (Rupp 2007) also known as MASTeR (Mustergultige Anforderungen - die SOPHIST Templates fur Requirements) (Sophist 2014) has been accepted as a standard for the syntactic specification of system requirements. This template has been recognized as a valuable aid tool so that the requirements are more precise and

have a standard syntactic structure that facilitates their understanding (Rupp 2007). However, anyone who has used the Rupp template in real projects has realized that some requirements cannot be expressed with that structure without some degree of ambiguity or inconsistency. That is the reason this article focuses on investigating the following research question: **What are the gaps that requirements engineers find when writing requirements in natural language and how to fill those gaps?** To find an answer to this research question, we have designed an experiment inspired by the *action science* (or *action research*) research method (O'Brien 2001). Two cycles of this method were conducted to analyze the requirements of two independent industrial projects. The first cycle of this action research method was reported in (Mazo and Jaramillo 2019) and the resulting template was used as input for the second cycle, which was oriented to requirements specifications for self-adaptive systems and represents an improved version of the Mazo and Jaramillo template, using the RELAX language (Whittle et al. 2009) as a reference in this cycle. Thus, with this research we aim to *analyze* the Rupp template *in order to* (i) evaluate their ability to represent industrial product requirements in a semi-structured way, and (ii) propose possible improvements to the template; *from the point of view of* two academics and two experienced requirements engineers *in the context of* two technology-based companies.

This paper is an extension of our previous work that appeared at CiBSE'19 (Mazo and Jaramillo 2019). In this paper, we significantly extended and improved the conference paper. First, we significantly extended the empirical study by evaluating our approach with one more real industrial project. Second, we introduce the implementation of the resulting template in the VariaMos tool (Mazo et al. 2015). Finally, we enriched the related work in this version.

The work resulting from this research is an adaptable and extensible template for specifying requirements of different domains (application systems, software product lines, cyber-physical systems, self-adapting systems). In the future, the template will be adapted and improved to address more domains.

This article is structured as follows: Section 2 explains the Rupp template; Section 3 describes the research method used for the experiment; Section 4 presents, using some examples, the most evident problems identified when using with the Rupp template; Section 5 presents the proposed improved template; Section 6 presents the preliminary evaluation of the new template; Section 7 presents the threats of validity of our study. Section 8 presents other initiatives specification templates for individual requirements and some related works; and Section 9 finally describes the conclusions and future work related to this research.

2 The syntactic structure of the Rupp template

As shown in **Figure 1**, the Rupp template consists of six spaces (denoted with A, B, C, D, E and F letters) to compose the syntax of a requirement. This section briefly describes each space of the template.

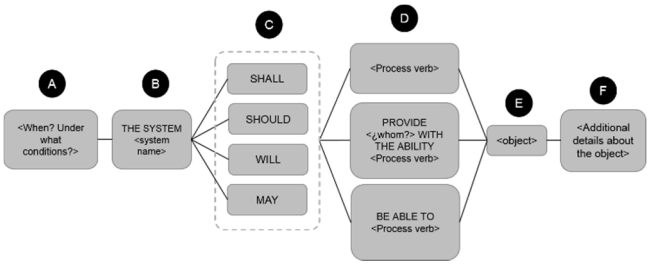

Figure 1. Rupp template.

(A) Conditions: The first space is a condition or a set of conditions, usually optional, at the beginning of the requirement. A condition can be logical: composed by the conjunction "If"; or temporary: composed by the conjunction "as soon as" or "after that".

(B) The System: The second space is the name of the system, the subsystem or component of the system that is specified for the requirement.

(C) Degree of obligation: The third space establishes the degree of obligation that the requirement can acquire. The template establishes four levels of obligation nature.

- The mandatory requirements, using the verb "shall"
- The recommended requirements, using the verb "should"
- The future requirements, using the modal verb "will"
- The desirable requirements, using the verb "may"

(D) Functional activity: The fourth space characterizes the functional activity that the system can assume, which includes the process verb object of the requirement. There are three types of activities:

- Autonomous requirement of the system: Indicates a functionality that the system performs independently without the need for interaction with users.
- User interaction: Indicates a functionality that the system provides to users.
- Interface requirement: Indicates a functionality that the system performs to react to events with other systems.

(E) Object: The fifth space is the object for which the behavior specified in the requirement is performed.

(F) Object details: The sixth and last space corresponds to the additional details (optional) about the object, the adjectives that qualify it or the characteristics that the object can possess.

Some examples proposed by Rupp (2007) for the specification of requirements with this template are the following:

- *The system should check whether the guest is registered*.
- *After the guest has selected the function "Place order", the system shall display the menu to the guest*.

- *The system shall provide the guest with the ability to place his order.*
- *If the chef has rejected the guest's order, the system should ask the guest whether the guest would like to choose another dish.*

The Requirements Engineering Magazine[1] presents some industrial cases in which the Rupp template was used.

3 Research method

The investigation reported in this paper was carried out through the research method called *action research* (O'Brien 2001). Action research is defined as "the intervention in a social situation in order to improve this situation and learn from it" (Wieringa and Morali 2012) (Susman and Evered 1978). The action research method aims to improve the practice by solving real problems and is conducted in order to investigate current phenomena in their natural context (Koshy et al. 2010). We have chosen this method because it allows us to answer the research question and achieve the objective of this research from an empirical experiment in an industrial context. Besides, (i) this research method can be executed at low cost since researchers play an active role in it; and (ii) the rigor of the action research method allows to reduce the threats to the validity of the experiment.

Susman (Susman 1983) developed a detailed model of the action research method with the five stages that must be carried out in each cycle of the process: diagnosing, action planning, taking action, evaluation and specifying learning. In the **diagnosing** stage, researchers identify the problem and collect the data required to carry out a detailed diagnosis. The **action planning** stage aims to define the different possible solutions that address the problem defined in the first step. During the **taking action** stage, a solution should be chosen and implemented. In the **evaluating** stage, researchers should analyze the data corresponding to the results of the chosen action plan. Finally, during the **specifying learning** stage, researchers should interpret the results of the action plan execution and learn according to the success or failure of the solution. Therefore, the problem is re-evaluated and a new cycle begins until the problem is solved and the stakeholders are satisfied with the obtained result.

To answer the research question, we carried out two cycles of the action research method as presented in **Figure 2**.

In this experiment, each cycle corresponds to the analysis of a form of specification of the requirements for two industrial projects. The experiment was carried out as follows. In the *first cycle*, we analyzed the requirements specification of the PeopleQA system of the SQA S.A. Company. PeopleQA is a system for human resource management, which facilitates the self-management of employees in different corporate activities such as permissions, vacation, performance measurement, and internal relations. Through the PeopleQA system, we proposed the first version of the new template to specify requirements in semi-structured NL. In this cycle, three possible solutions were analyzed: prose style requirements specification (as the stakeholders expressed them), specification using the Rupp template and requirements specification using an improved version of the Rupp template that we call the Mazo & Jaramillo template.

In the *second cycle* we analyzed the requirements specification of the Yuke-GreenHouse System of the Koral Company, Yuke-GreenHouse is a self-adaptive system for controlling irrigation, temperature, and environment in greenhouses and coffee crops in Colombia. In the second cycle, three possible solutions were analyzed: prose style requirements specification, specification using the Mazo & Jaramillo template, and requirements specification using the new improved template presented in this paper.

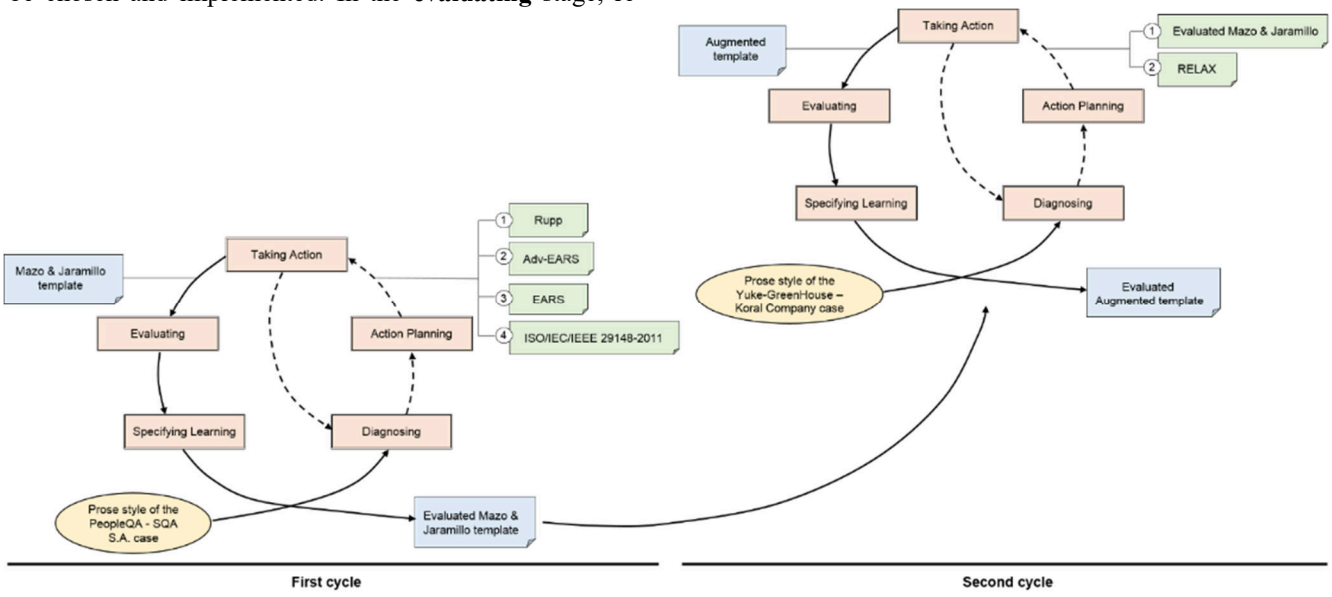

Figure 2. Research process

[1] RE Magazine (https://re-magazine.ireb.org/)

In each cycle the following stages were executed:

1. *Diagnosing:* Some problems were identified when using prose style and the Rupp template to write the requirements of the first case, and when using prose style and the Mazo & Jaramillo template to write the requirements of the second industrial case. This stage was conducted through several mini-cycles of requirements specification in order to identify the problems associated with this activity and to collect the information needed to create the new template proposed in each cycle and to achieve a systematic response to the research question.
2. *Action Planning:* Templates of requirements proposed by other authors were considered. In each cycle, it was evaluated that the improved template (resulting from each cycle) was consistent with other than the Rupp template, we considered other templates such as EARS (Mavin et al. 2009), Adv-EARS (Majumdar et al. 2011a) (Majumdar et al. 2011b) and ISO/IEC/IEEE 29148-2011 (ISO/IEC/IEEE 2011). To ensure that the improved template produced in the second cycle remained consistent with the considered templates, we planned and executed the following strategy: At the beginning of each cycle of requirements writing, the templates found in the literature (not all of them were found from the first cycle) were used as inspiration artifacts to incorporate their relevant elements in the new template produced at each cycle. Thanks to this strategy it was possible to improve our baseline templates (i.e., the Rupp template in the first cycle and the Mazo & Jaramillo template in the second cycle) in the situations where this template was not adequate.
3. *Taking Action:* In this stage, we first considered the requirements that could not be fully specified using the reference templates of each cycle. For these requirements, we evaluated to what extent they could be syntactically specified using the templates found during stage 2. We performed this evaluation in order to find requirements specification reproducible patterns. Every time that a reproducible pattern was identified in at least three requirements with similar conditions, this pattern was added to the new template proposed in each cycle in order to enrich them.
4. *Evaluating:* At the end of each cycle, it was evaluated whether the proposed template allowed to specify at least 98% of the industrial case requirements corresponding to the current cycle. The main criteria to evaluate the representation of requirements is that they do not present problems of ambiguity, vagueness, complexity, omission, duplication, verbosity, non-implementation and untestability. Mavin *et al.* (Mavin et al. 2009) and (Rupp 2007) give us a more detailed description of these criteria, which are considered a *de facto* standard in requirements engineering.
5. *Specifying Learning:* At the end of each cycle, the authors interpreted of the results obtained. Then, based on these results they determined the strengths and limitations of the improved template produced in each cycle.

The various phases and the succession of cycles are collaborative since the research process and objective have been carried out in collaboration between the authors. This is another characteristic that led us to choose action research as a research method for this work. The research process consists of two cycles, one for each industrial case we had at our disposal. Although two cases are not enough to propose a generic set of extensions for the Rupp template, the second case provides supplementary evidence that allowed us to re-evaluate and improve the template we reported in the previous version of the article. The use of new real cases to evaluate an engineering artifact in its early stages is welcome and usual in empirical research processes such as the one reported in this article. We, therefore, hope that this new template will be evaluated in many more cycles with new and varied industrial cases that help to collectively build the RE template that the industry requires.

4 Problems identified in the baseline templates

4.1 First cycle

The prose style requirements specification corresponding to the PeopleQA system of the SQA S.A. Company was rewritten with five requirements specification templates as presented in **Figure 2**. The use of each template corresponds to a micro-cycle into the first cycle of the action research process. At the end of these micro-cycles, we produced the first version of the Mazo & Jaramillo template that was then evaluated and improved in the subsequent two stages of the first cycle.

The problems and gaps detected when working with the templates considered in these micro-cycles are described below. These problems and gaps were saved in a document, available online[2], which contains each of the requirements of the industrial case and each of the problems encountered during the investigation. In particular, the first sheet presents the requirements in prose style; the second sheet presents the requirements using the Rupp template; the third sheet summarizes the problems identified when using the Rupp template; and the fourth and last sheet presents the requirements specified with the constructs borrowed from other templates found in the literature. For each of these types of problems, we have defined a descriptive name, a brief description and an example to better understand the problem.

Missing reasons

Sometimes it is necessary to express the reason for a requirement. For example, in agile development frameworks, one of the most important aspects in the specification of requirements through user stories is to specify the "why" or the "for what" of the requirement (Cohn 2004) (Beck 1999). This gives a better context to who implements the functionality or

[2] Requirements specification 1st cycle - (http://shorturl.at/cpDEO)

behavior that describes the requirement and will allow him to better understand the level of importance or priority of the requirement. For example, the requirements: *The VMS (Vital Monitoring System) must have the ability to interact with other devices of nearby people to know their vital activity.* And *If any sensor exceeds the defined tolerable limits, the home automation system must light a siren to warn the homeowner.* have a "for what" of vital importance because both requirements belong to critical systems. If a requirements specification template allows defining the reason for the requirements, developers can easily understand it, because it is explicitly stated how important is to implement those requirements with high-quality levels.

Omission of quantities and ranges

Sometimes the requirements refer not only to a specific object but to several objects or a range of objectives of the same nature. Some of the analyzed templates (e.g., the Rupp template) do not explicitly allow the possibility of specifying ranges or quantities of objects in the requirements. As presented in the following example, the omission of an amount would have led to ambiguities or inaccuracies: *The point of sale subsystem must provide the POS administrator with the ability to link between one and maximum 10 warehouses at a point of sale*.

Omission of biconditionals

Some requirements require certain behaviors performed only if certain conditions are met; otherwise, the behavior cannot be performed. We call this *biconditional* to express that behavior *A* is performed "if and only if" behavior *B* is fulfilled and vice versa. For example, in the requirement: *The point of sale subsystem must show the boxes if and only if they are in the active state*, the "*show the boxes"* behavior will be performed only for objects that are in a certain state and not for all objects within the domain. Here there is an explicit condition that the requirement must effect through the process verb "show", using the conditional "if and only if". Consider another example of a requirement: *After the vacuum has been turned on, the Ivaccum system should start the cleaning cycle if and only if the vacuum's battery charge is 90% or more*. In this case, the behavior of the object depends on a condition on the charge of the battery.

As can be seen, these types of conditions are common when specifying requirements in industrial cases; however, some of the analyzed templates (e.g., the Rupp template) do not explicitly provide a way to express this kind of specifications.

Gap in conditionals

Requirements behaviors are conditioned by different factors, which imply different interpretations depending on these conditions. For example, a requirement that specifies *While the temperature control is on, the system must balance the ambient temperature* can have a different interpretation to the requirement that specifies *If the temperature control is on, the system must balance the ambient temperature* and also both can be differentiated from a requirement that specifies *As soon as the temperature control is turned on, the system must balance the ambient temperature*. In all three cases, although a similar condition is used, the interpretation is different. In the Rupp template, only two types of conditionals are used, which are: the logical conditionals and the temporary conditionals (Rupp 2007). However, we found other types of conditions in the rest of templates, for example, for behaviors that are triggered by events and for behaviors that take place while the system is in a certain state.

Lack of verifiability of non-functional requirements

Some of the templates analyzed in the first cycle were created to specify functional requirements. Thus, explicit structure for the adequate writing of measurable and finite factors to define the satisfaction (level) of non-functional requirements and restrictions was a recurrent weakness of the templates analyzed during the first cycle. For example, these two quality requirements: *The system should be available 7x24x364 for users* and *The performance of the system must be optimal, trying to respond to users in less than two seconds* have a measurable and finite factor to determine that the requirement will or not satisfy the need of the interested parties.

Lack of reference to external systems or devices

In case the type of system activity is an interface requirement, the syntactic structure of some of the analyzed templates does not explicitly refer to external systems or devices. For example, the requirements *The point of sale subsystem must be able to read bar codes on item labels* and *The system should be able to obtain the information of a client* follow the syntactic structure proposed by the Rupp template; however, none of these requirements mention the name of the system or device with which information is exchanged, nor it is established if the information goes to or from the device or system.

Lack of concepts to write domain requirements

In some cases, the requirements do not refer to a product but several products of the same family (Mazo 2018a). Product lines are based on the concept of variability management to specify, design and intensively develop the products of the same family in a prescribed manner. Although some of the analyzed templates can be used to specify requirements with different priority levels, they cannot be used to specify their variability. For example, in the requirements: *The product line of virtual stores must calculate the VAT value of each purchase* and *The product line of virtual stores could calculate the VAT value of each purchase* two levels of priority are specified, but the variability of the requirements is not considered. Indeed, it is not said if it is for all products of the product line (mandatory for all the products) or only for some of them (optional).

4.2 Second cycle

The requirements specification of the Yuke-GreenHouse case written in prose style was rewritten with two templates. The first template used in this second cycle is the one produced in the first cycle and the second one corresponds to the RELAX language (Whittle et al. 2009). Each rewriting of the requirements of the Yuke-GreenHouse case with those two artifacts corresponds to a micro-cycle into the second cycle of the action research process as presented in **Figure 2**. At the end of these two micro-cycles, we produced the Evaluated new template that was then evaluated and improved in the subsequent two stages of the second cycle.

The problems and gaps detected when working with the artifacts considered in these micro-cycles are described below and available online[3]. For each of these types of problems, we have defined a descriptive name, a brief description and an example to better understand the problem.

Lack of concepts to write requirements for self-adaptive systems

Self-adaptive systems have the ability to autonomously modify their behavior at runtime in response to environmental and changing system conditions. Self-adaptation is particularly necessary for applications that must be executed continuously, even in adverse conditions and with changing requirements (Whittle et al. 2009). In general, self-adaptive systems include automotive systems, telecommunication systems, environmental monitoring, and smart home systems. The main problem faced by requirements engineers is that the typical behaviors of this type of system can vary due to environmental uncertainty conditions, caused by multiple reasons such as weather, sensor failures, unexpected conditions, the variability of data, among others.

Inability to manage uncertainty

Uncertainty is one of the characteristics of self-adaptive systems, therefore this type of requirement must ensure that the system meets the needs of the stakeholders while at the same time adapting to the conditions of the environment. Thus, the satisfaction of these requirements should be defined with satisfaction at some level on a continuous scale defined by a fuzzy function (Jureta et al. 2015). The Mazo & Jaramillo template does not consider the uncertainty for self-adaptive requirements. For example, a requirement that specifies: *If the ambient temperature rises above 25 degrees, then the self-adaptive system Oktupus must raise the temperature level to 30°* establishes an invariant restriction (Whittle et al. 2009) that make it difficult to adapt the system to certain environment variables.

Lack of specificity in temporality

Self-Adaptive systems use timing functions and frequencies to adapt themselves to the environment. Handling these aspects is also a weakness of the Mazo & Jaramillo template. Let's consider the following requirement: *The Oktupus self-adaptive system must measure the temperature of the room every hour*. In this case, it would be desirable to be able to relax the requirement to better adapt the measurement period to also consider the changing conditions. This would imply that the system would be able to measure the temperature not only every hour but also every time there is a major change in the system.

5 Proposing a new requirements specification template

Considering each of the problems encountered during the execution of the action research method and exemplified in Section 4, then we have improved the Rupp template (Rupp 2007) and subsequently the Mazo & Jaramillo template (Mazo and Jaramillo 2019). The Mazo & Jaramillo template (c.f. **Figure 3**) was created as a result of the first action research cycle and is composed of eight spaces. Each space was structured thinking a simple and robust syntactic specification to cover the most types of requirements in several types of systems. The rectangles in *yellow* represent conditionals; *gray* rectangles are used to represent the family of systems, the system or a part of it; the *orange* rectangles represent the degree of obligation; the *green* rectangles are the

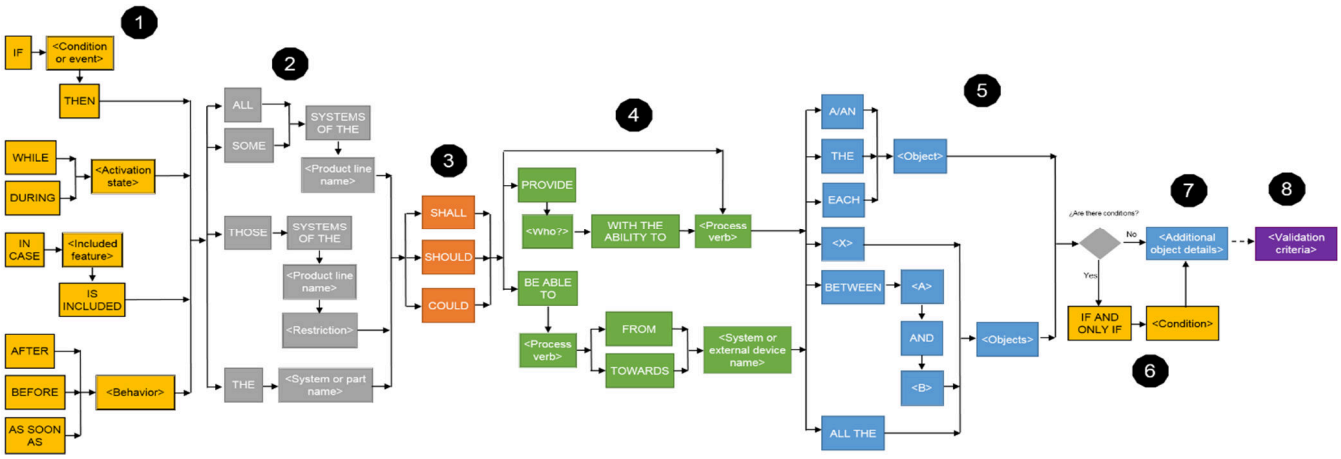

Figure 3. Mazo & Jaramillo template.

[3] Requirements specification 2nd cycle - (http://shorturl.at/cpDEO)

activities characterizing the system; the *blue* rectangles represent the objects (nouns), with their respective quantities and complements; and the *purple* rectangle describes the measurable criterion of verification of the requirement. The latter is optional, for that reason is represented through a dotted line. The improvement made to the Mazo & Jaramillo template is inspired by concepts from other related works found in the literature, e.g. the EARS template (Mavin et al. 2009), which establishes a set of syntactic rules for the specification of requirements through the use of conditional clauses that trigger functional behaviors and described in the RELAX requirements language (Whittle et al. 2009), which incorporates various types of operators to address the uncertainty in the behavior of a self-adaptive system. Thus, the new requirements specification template proposed in this paper is presented in **Figure 4** and it is the result of the second action research cycle that follows the first research cycle reported in the CIbSE conference (Mazo and Jaramillo, 2019).

Templates for user requirements specifications, such as Connextra for writing user stories (Davies 2001), were not considered in this article because our template is oriented to the specification of system and software requirements, while user stories are oriented to the stakeholders (Wiegers and Beatty 2013). Templates oriented to user requirements specification are beyond the scope of this article.

In the remainder of this section, we describe each of the components of the resulting template at the end of the two action research cycles.

5.1 Conditions under which a behavior occurs

Some requirements do not describe continuous behaviors, but behaviors that are performed or provided only under certain conditions; for example, logical or temporary, as is shown below.

a. Requirements with logical conditions. They are used for describing behaviors that are triggered only when a logical condition is met (Rupp 2007) or when an unexpected event occurs (Mavin et al. 2009). The form is:

   IF <Condition or event> **THEN** (ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD

   For example: *If the number of products in a warehouse reach the defined minimum limit then, the inventory subsystem should generate a product replacement alert for that warehouse.*

b. Requirements guided by the state. They are used for describing behavior that must be performed in the system while the system is in a specific state. This condition was proposed by (Mavin, et al. 2009). The form of this specification is:

   WHILE|**DURING** <Activation state> (ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD

   For example: *While the payment of an invoice from a customer has not been confirmed, the subsystem must send a daily text message to the cell phone number registered by the customer.*

c. Requirements with optional elements. They are used for describing behavior that must be performed only if a particular characteristic is included (Mavin, et al. 2009). The form of this is specification is:

   IN CASE <Included feature> **IS INCLUDED** (ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD

   This condition is especially useful in domain requirements when you want to incorporate certain requirements depending on the characteristics provided by the product line.

   For example: *In case the text entry action is included, all systems of the test automation framework product line shall provide the tester with the ability to enter a specific text, in a form field.*

d. Requirements with temporary conditions. They are used for describing behavior that must occur after another behavior occurs. They occur sequentially, it means, behavior *A* is done after *B*. This condition was proposed by (Rupp 2007). The form is:

   AFTER|**BEFORE**|**AS SOON AS** <Behavior> (ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD

   AFTER means that the system must have completed a running behavior before initiating another behavior. **BEFORE** means that the system must initiate a behavior before another behavior takes place. **AS SOON AS** means that the system does not necessarily have to have finished a running behavior before initiating another behavior.

   For example: *After reading the products for a particular location, the Inventory subsystem should provide the warehouse owner with the ability to close the product count for that location*.

e. Requirements with complex conditions: For requirements with more complex conditional clauses, it can be necessary to add with keywords as *When*, *While*, *Where*. The keywords can be integrated into more complex expressions to specify richer behaviors of the system (Mavin et al. 2009). As expressed in the following example:

   When a cash settlement operation is performed on a cash register, while the box is temporarily closed, the point of sale subsystem should show the amount of cash that is in the box.

   Conditional clauses can also be structured using the Boolean operators *AND*, *OR* and combined with *NOT* (Rupp 2014). For example:

   If a location contains products and the option to delete a location has been selected, then the Inventory Subsystem should display an alert message indicating that the selected location cannot be deleted.

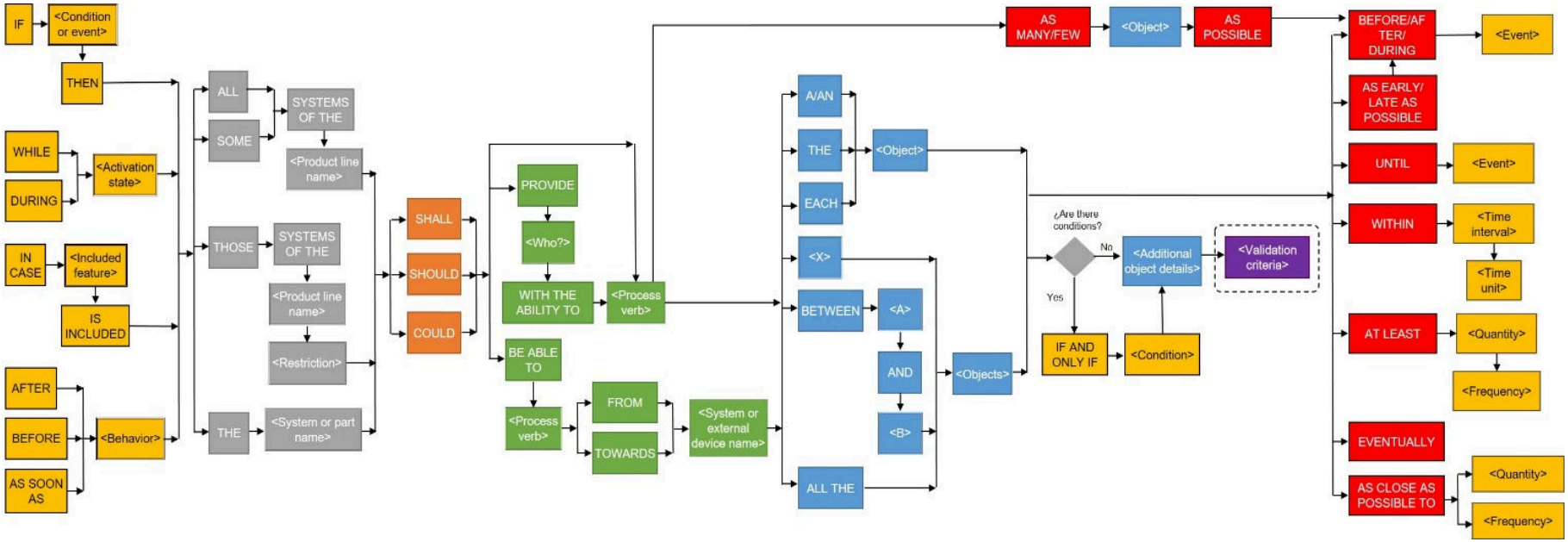

Figure 4. New template for the specification of requirements in semi-structured natural language.

The requirements guided by the state and the requirements with optional characteristics were taken from EARS (Mavin et al. 2009), and the requirements with logical and temporal conditions were taken from the Rupp template (Rupp 2007).

5.2 Family of systems, systems or parts of a system

This space in the template is reserved for the name of the product line, system, subsystem or system component. In the case of a product line requirement, it must be specified whether the requirement is valid for all or only for some systems.

We completed the second space of the Rupp template (cf. B space in **Figure 1**) with the possibility of specifying product line requirements since this template was not correctly adapted to be able to write them in semi-structured NL. The structure of the second space of the new template is as follows:

ALL|**SOME** SYSTEMS OF THE <Product line name>

In some cases, we must consider certain behaviors that some systems of the product line must incorporate if certain conditions or restrictions are met when this happens, we will use the expression:

THOSE SYSTEMS OF THE <Product line name> <Restriction>

Some examples of product line requirements, using the improved template are:

In case the action of comparing text is included, those systems of the automation framework product line that only include the option to enter text shall provide the tester with the ability to configure a text for comparison with another element.

If the automation framework is web-based, all systems of the Test Automation product line shall provide the tester with the ability to select the type of browser where the test will be run (be it Chrome, Firefox or Safari).

5.3 The degree of priority

In the Rupp template, this space (cf. C space in **Figure 1**) is traditionally reserved to specify the degree of obligatory nature of the requirement; however, we changed the "obligatory" concept to the "priority" concept in order to not confuse it with the "mandatory" concept of product lines. To define the priority of the requirements we have used the MoSCoW technique (Clegg and Barker 1994), in which three degrees of priority are established: essential, recommended and desirable.

a. Essential requirements. These requirements must be implemented to achieve the success of the product or the product line. The word **SHALL** is used.
b. Recommended requirements. These requirements are important, but not necessary to achieve the success of the product or the product line. The word **SHOULD** is used.
c. Desirable requirements: These requirements are desirable, but not necessary. They could improve the user experience and customer satisfaction. The word **COULD** is used.

Some examples of requirements with differentiation of the degree of priority, using the improved template are:

All systems of the Test Automation product line shall incorporate a click action.

If a motion sensor is activated, then Oktupus system should send an instant image to the home owner's email.

5.4 The activity

The fourth space, the same as the Rupp template (D in **Figure 1**), specifies the characterization of the activity that is conducted by the system or by the systems of the corresponding line. There are three types of activities that can be performed:

a. Autonomous activity. In this kind of activities there is no user involved, which means that the (sub) system or systems initiate and execute the behavior autonomously. The form of this type of activity is:

```
ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD <Process verb>
```

b. User interaction. In this activity, the (sub) system or systems provide a user with the ability to use certain behavior that is initiated or stimulated by a user (actor) that interacts with the system(s). The form of this part is:

```
ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD PROVIDE <Who?> WITH THE ABILITY TO <Process verb>
```

Where **Who** is the actor or user that should have the ability to use the functionality. The user must be correctly characterized and not incur the undue use of nouns without a reference index (Rupp 2007); it means, indicating "the user" would be an error that would lead to an ambiguity in the specification.

c. Interface requirement. In this activity, the system performs a behavior dependent on another entity (which can be another system or a physical device). This space was improved in the new template by explicitly adding the name of the external entity with which the system interacts and the direction of the relationship. The form of this type of activity is:

```
ALL|SOME SYSTEMS OF THE <Product line name>)|(THE <System or part name>) SHALL|SHOULD|COULD BE ABLE TO <Process verb>
```

In addition, this structure is completed with the entity with which the system interacts:

- If the behavior is executed by the external system that transmits data to the receiving system interface, then the specification will be complemented by adding:

  FROM <System or external device name>

- If the behavior is performed by the system and interacts or affects another system or external device then the specification will be complemented by adding:

TOWARDS <System or external device name>

An example in this case for an interface requirement is:

The point of sale subsystem shall have the ability to read a valid credit card from a branch's dataphone

5.5 The object or objects

This space is reserved for the object or objects that make up the system. In the new template, we have incorporated the concept of *range*, since the objects can be affected in different ranges. The ranges in the new template are specified as follows:

a. Single object: **ONE** <Object>
b. A specific object: **THE** <Object>
c. Each object of a set: **EACH** <Object>
d. Multiple objects: <X> <objects>, where X is the number of objects
e. Range of objects: **BETWEEN** <A> **AND** <B> <Objects>, where A is the lower range and B is the upper range
f. All objects in a set: **ALL THE** <Objects>

Two examples of requirements with ranges of objects, using the improved template are presented as follow:

The inventory subsystem should provide the inventory manager with the ability to associate between 1 and 3 bar code reading guns to a cash register.

As soon as the daily activity cycle ends, the Oktupus system must restart all the sensors connected in the home.

5.6 The complementary details

In the sixth space of the template, complementary details of the object are specified. They can be one or several adjectives, as well as a more enriched description of the object, without the risk of altering the proper meaning of the specification of the requirement and focusing only on describing the details related to the object in question. This template space was retained from the Rupp template and was not modified by the authors.

5.7 Conditionality in the object

Sometimes, the behavior of the requirement is conditioned by the state of an object. In the new template, we have reserved the seventh space to specify a behavior that the system must carry out if and only if the object meets a certain condition. In this case, the requirement is completed by adding the following expression:

IF AND ONLY IF <condition>.

It is important to clarify that this condition is optional. It is only given explicitly if the precise object of the requirement requires specifying the condition, therefore, it is not mandatory in the specification of the requirement.

Here are two examples of requirements with conditionality in the object, using the improved template:

If any sensor exceeds the defined tolerable limits, the Oktupus system should turn on a siren, if and only if the siren is activated.

The inventory subsystem could provide the warehouse manager with the ability to eliminate a purchase order, if and only if the purchase order has not been dispatched.

5.8 Verification criterion (adjustment) of the requirement

In some types of requirements, especially non-functional requirements, it is necessary to establish the degree to which the requirement must be met. Robertson and Robertson (Robertson and Robertson 2013) suggested including adjustment criteria; it means including "a quantification of the requirement that demonstrates the standard that the product must reach" as part of the specification of each requirement, functional and non-functional. The adjustment criteria describe a measurable way to assess whether each requirement has been successfully met.

For this purpose, in the new template, we have added, in the last space, the option of establishing a measurable or observable criterion to determine the degree of verifiability of the requirement. This was done to make sure that the requirement can be verified either by a person or a machine. This criterion is defined at the discretion of the author of the requirement and is optional, although it is recommended to always use it in the quality requirements.

Here are two examples of non-functional requirements using the improved template:

If a fault causes the system to stop, the Oktupus system must restart all the sensors in less than 20 seconds.

The system must provide an ATM with the ability to register a sale in a cash register without presenting more than 2 different screens.

5.9 Relax requirements statements for self-adaptive systems

To deal with problems of environmental uncertainty (see Section 4.8) of self-adaptive systems, especially in the requirements specification phase, (Whittle et al. 2009) propose a language called RELAX. RELAX incorporated several types of operators to address the uncertainty in the properties of the system. Usually, requirements prescribe the behavior using imperatives such as "Must" or "Should". These imperatives define the functional behavior that a system should always provide. However, for self-adaptive systems, environmental uncertainty may mean that it is not always possible to achieve all the "Must" statements. Therefore, it may be necessary to make concessions among "Must" statements in order to make some non-critical behaviors more flexible in favor of more critical ones (Whittle, et al. 2009). RELAX proposes to establish a simple process to explicitly identify when a requirement should remain unchanged and mandatory and when a requirement can temporarily relax under certain conditions. Although RELAX is not a specification template, in this article we have employed several operators

proposed by RELAX to make it easier to specify requirements in self-adaptive systems.

Figure 4 presents the proposed structure of the improved template for the specification of self-adaptive requirements. Although this template is based on RELAX, we also incorporated some ideas, which were inspired by the works presented in (Baresi, et al. 2010). In particular, we incorporated (i) fuzzy conditions that can be taken by self-adaptive systems to measure different environment variables (Souza, et al. 2011); (ii) Awareness Requirements (or AwReqs) to specify requirements about the success or failure of other requirements that can refer to goals, tasks, quality constraints, and domain assumptions and (iii) constraints that must be met using certain questions in a conditional manner (Ibrahim et al. 2014). These concepts completed the Mazo & Jaramillo template allowing to specify requirements of self-adaptive systems.

Due to the autonomous nature of the requirements for this type of system, we have implemented an equivalent template for requirements in self-adaptive systems. This space of the template can be identified in red color and must be written in the requirements, at the end of their textual specification. For this version of the improved template we have omitted the types of activity (see Section 5.2) interaction with users (user interaction requirements) and interaction with other systems or devices (interface requirements), we also omit the conditionals of objects and verification criterion, which are explicit in each of the operators to "relax" requirements.

In the remainder of this sub-section, we explain each part of the red space of the improved template.

a. (`AS MANY|AS FEW) AS POSSIBLE`: A requirement must maximize or minimize a certain occurrence of something or a certain amount of objects, as many or as few as possible, thus leading to adaptation.
b. `BEFORE|AFTER|DURING`: A requirement must be met before, during or after a particular event, usually,i these three operators go after the operators **AS MANY AS POSSIBLE**, **AS FEW AS POSSIBLE**, **AS SOON AS POSSIBLE** or **AS LATE AS POSSIBLE**.
c. (`AS EARLY|AS LATE) AS POSSIBLE`: A requirement specifies something that must be fulfilled as early as possible or must be delayed as late as possible.
d. `UNTIL`: A requirement must be maintained until a future position (event).
e. `WITHIN`: A requirement must be maintained for a particular time interval, expressed in units of time.
f. `AT LEAST`: A requirement must meet a minimum frequency or time, until infinity.
g. `EVENTUALLY`: The behavior of the requirement must occur eventually, e.g. it is not completely safe or fixed that the behavior occurs, but the system must be prepared.
h. `AS CLOSE AS POSSIBLE TO`: A requirement specifies something that happens repeatedly, but the frequency can be flexible (above or below the specified frequency, but as close as possible to this value) or a requirement specifies an amount (quantity), but the exact amount can be flexible (above or below the specified amount, but as close as possible to this value).

Below we present some examples of a VMS (self-adaptive Vital Monitoring System) case using an intelligent bracelet and we use the improved template to specify these requirements.

- *The VMS system must record as many steps as possible during a user's walking activity.*
- *The VMS system will consume as few units of energy as possible during the normal operation of the intelligent bracelet.*
- *The VMS system must send an alert to the user who must stop when the physical activity levels (MET[4]) are as close as possible to 2.*
- *If a user's vital signs levels are below the user-defined values, then the VMS system must send an alert to the registered emergency phone within 2 seconds.*
- *The VMS system should check the user's average calories consumed levels eventually.*

6 Preliminary evaluation of the proposed template

As mentioned in Section 3, two action research cycles were performed for this research. For the preliminary evaluation of the template proposed in this article, two groups were established in each action research cycle. Each group was made up of two roles: a business analyst with similar experience (that is, the same number of years in the company and participation in comparable projects in the subject, duration and size) in requirements engineering; and a technical requirements reviewer.

In the first action research cycle, the analyst of each group had to specify the requirements of the PeopleQA system of the SQA S.A. Company within the Rupp template (for the first group) and within the Mazo & Jaramillo template (for the second group) and the second role inspected the work of each group.

The requirements specification document of the PeopleQA system corresponds to 46 requirements in prose style. Then, the first group of business analysts rewrote the requirements using the Rupp template and with the accompaniment of the authors of this paper to support them in resolving doubts. Through this method, requirements were

[4] Measurement of physical activity according to the World Health Organization (https://www.who.int/dietphysicalactivity/physical_activity_intensity/en/)

identified and specified. Once the specification was concluded, the technical reviewer of the first group identified 48 requirements with specification problems that do not adhere to the standard proposed by Rupp, meaning that 34.8% of the requirements had problems of adherence with the Rupp template. These problems were categorized according to seven types of criteria as shown in **Table 1** (for a more detailed explanation of each of these types of problems and some examples that may be presented in a requirements specification based on the Rupp template see Section 4). The second group of business analysts focused on specifying the prose style requirements of the PeopleQA system within a new template that was built and incrementally improved through four mico-cycles (each one corresponding to the rest of templates used in the first action research cycle as presented in **Figure 2**). The results of specifying the requirements of the PeopleQA system within the resulting template were interesting. Thus, it was possible to successfully specify 98% of the requirements using the template proposed in this article. Only three requirements could not be fully specified using the improved template, but it is noteworthy that 135 achieved a specification that adheres to the improved template, without any observation by the technical reviewer.

The main factor by which some requirements continue with some problems when using the improved template is because it was detected in the first cycle that some requirements when they have a restrictive behavior, that is, when the requirement specifies what the system should not do, instead of what you should do, the template does not adhere properly. According to (Wiegers and Beatty 2013) these types of requirements are known as negative requirements and will be part of a later investigation on how to specify this type of requirement.

Table 1. Problems identified in the requirements in the first cycle.

Identified problem in the first cycle	# of requirements using the Rupp template	# of requirements using the proposed template
Inappropriate conditionality	6	0
Lack of reference to external systems or devices	4	0
Omission of biconditionals	7	0
Omission of quantities and ranges	18	0
Lack of verifiability of non-functional requirements	3	0
Missing Reasons	3	0
Others	7	3

In a second cycle, for the requirements specification of the Yuke-GreenHouse system of the Koral Company, a self-adaptive system of an intelligent nursery called Yuke-GreenHouse, 12 requirements in prose style were obtained. There prose style requirements were represented as 46 requirements within the Mazo & Jaramillo template. It is important to consider that of these 46 requirements only 22 fulfilled the conditions to be considered self-adaptation requirements. Evidently, these 22 self-adaptation requirements had adherence problems with the Rupp template because Mazo & Jaramillo template is an extension of that template. These problems are shown in **Table 2**.

Table 2. Problems identified in the requirements in the second cycle.

Identified problem in the second cycle	# of requirements using the Rupp template	# of requirements using the proposed template
Inappropriate conditionality	2	0
Lack of concepts to write requirements for self-adaptive systems	11	0
Omission of biconditionals	1	0
Omission of quantities and ranges	1	0
Lack of specificity in temporality	1	0
Inability to manage uncertainty	9	0

The Mazo & Jaramillo template was improved in the second action research cycle which leads to writing three additional requirements, thus completing 25 requirements. 13 of those 25 requirements were invariant; thus, according to (Whittle, et al. 2009) those are requirements that are strict in compliance and cannot be flexibilized. 12 of these 25 requirements had behaviors that reflected factors of uncertainty and can be "RELAX-ed"; therefore RELAX operators were applied for this type of requirement.

The improved template has been implemented to be used in the VariaMos tool (Mazo 2018b) in order to facilitate the writing of domain requirements, application requirements and self-adaptation requirements (for example, self-adaptable cyber-physical systems). VariaMos is available online[5] and through the RequireX option it is possible to access the forms that implement the template proposed in this paper.

Figure 5 illustrates an excerpt of the form for specifying domain requirements. **Figure 6** illustrates an excerpt of the form to specify self-adaptative requirements. **Figure 7** illustrates the administrative panel. This interface provides the ability to execute vital actions such as Create a new requirement, edit and delete, In addition, the ability to generate two types of reports, a general report by category and another for each requirement in pdf format. **Figure 8** illustrates an example of a generated requirements document.

[5] VariaMos – (www.variamos.com/variamosweb)

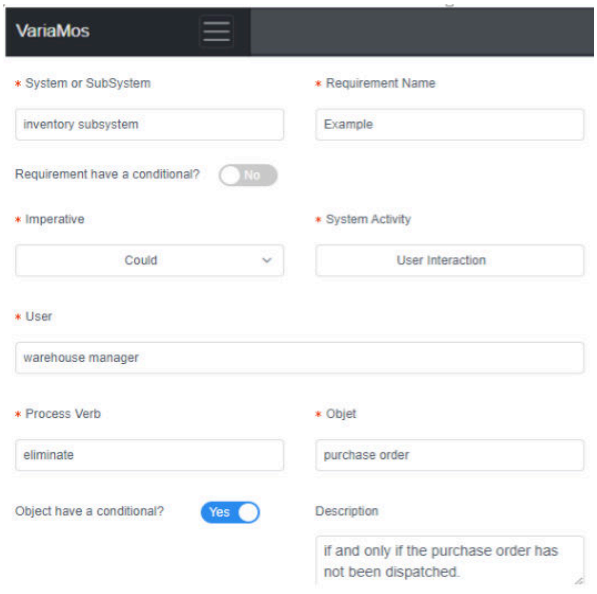

Figure 5. Domain requirements form.

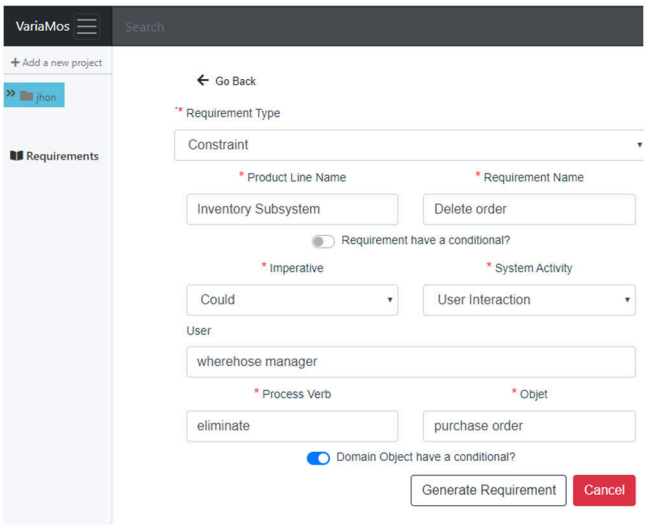

Figure 6. Self-adaptive requirements form.

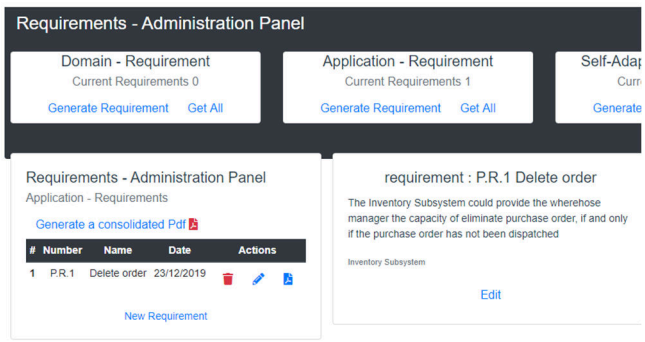

Figure 7. Administration panel.

Variamos – RequireX

RequireX - Application requirements

Id	System	Requirement
P.R.1	variamos	The variamos shall provide the admin the capacity of find all users
P.R.2	inventory subsystem	The Inventory subsystem could provide the warehouse manager the capacity of eliminate purchase order, if and only if the purchase order has not been dispatched.

Figure 8. Generated requirements document example.

7 Threats to validity

This section aims to demonstrate that the result of the experiment is valid for those in charge of writing the requirements, business analysts and requirements experts. We consider three types of threats to the validity (Cook and Campbell 1979) of the experiment: (i) the validity of the conclusion, (ii) the internal validity and (iii) the external validity.

7.1. Validity of the conclusion

Concerning the **statistical power of statistical tests**, the research-action method used in this experiment is exploratory and qualitative, not quantitative, and there is no statistical hypothesis test; therefore, the threat of low statistical power does not apply. According to **the threat of reliability of the implementation of the treatm**ent, in the experiment that served us to obtain an improved template of specification of requirements, we applied the treatments to each of the industrial cases in a **homogeneous** way starting with the Rupp template, using other templates and concepts found in the scientific literature, and our analysis to improve it iteratively. However, we are not sure that the results were the same if we had started from another template. We decided to start the experimentation with the Rupp template since it is the *de facto* standard and we wanted all its constructs to be present in the resulting template.

7.2. Internal validity

From the point of view of history, the two cycles of the experiment were carried out over a six months; in this period there were no relevant environmental, social or personal factors that affected, in one way or another, the results of the experiment. From the point of view of maturity, the authors established a specific scope for each system and stabilized the requirements specification document for each case to prevent it from changing between each cycle.

7.3. External validity

The interaction between selection and treatment can be a threat to the validity of the experiment and to avoid a biased result by a single case. We use two industrial cases, and two people to execute the treatments. This experiment involved experienced people in requirements elicitation and specification. Participants were also concerned by the specification of system requirements (programmers, testers, end-users, project managers).

8 Related work

Apart from the Rupp template, there are other templates proposals for the specification of requirements in semi-structured NL. Thus, EARS (Easy Approach to Requirements Syntax) (Mavin, et al. 2009) is one of the templates that considers several conditional patterns from which several requirements are typified. For example, *ubiquitous requirements* (they do not have a precondition that triggers behaviors, but they are always active), *event-driven requirements* (describe a behavior that occurs in the system when an event is triggered) and *guided by states* (describe a behavior that is active while the system is in a defined state). The EARS template focuses on conditional patterns under which the requirements are presented; additionally, EARS focuses on the aeronautical industry. Unlike this, our proposal is intended to be independent of the application domain (see Section 4) and aims to support the writing of functional and non-functional requirements and restrictions.

Alexander & Stevens (Alexander and Stevens 2002) propose a template for writing functional requirements from the user's perspective; since it is more natural to formulate the requirements in terms of the action of a user, not from the perspective of the system (Wiegers and Beatty 2013). The structure of the template is as follows:

```
The <User Type or Actor Name> has the ability to <Process Verb> <for some object> <Measure or rating criterion, response time or quality declaration>
```

Unlike the work proposed by Alexander and Stevens, our approach is oriented to the specification of the requirements from the perspective of the behavior of the systems and not of the needs of the interested parties.

Adv-EARS (*Advanced EARS)* (Adv-EARS 2011) proposes a syntax in semi-structured language to specify functional requirements, in such a way that automated support is given for the derivation of use cases and actors in use case models. Unlike Adv-EARS, our template focuses on functional and non-functional requirements, while Adv-EARS focuses solely on functional requirements. This syntax is an advanced version of EARS (Mavin et al. 2009), so some elements of Adv-EARS could be incorporated in our work in the future.

The CESAR research project, funded by the European Union's ARTEMIS program, reviewed the work on the use of templates (ARTEMIS 2010) intending to extend and apply the approach to several critical domains for security, with discussions on how to formalize the approach using ontologies.

In (Souza et al. 2011), the Awareness Requirements concept is introduced. This concept is related to other requirements and their success or failure evaluation at runtime. In this work, the importance of monitoring requirements at runtime to provide feedback loops is emphasized. This work, like the work proposed by (Whittle, et al. 2009), in which the RELAX language is proposed to write self-adaptation requirements, served as a frame of reference for this article.

Some other articles complement our work; for example, (Tjong, et al. 2006) and (Denger, et al. 2003) present two proposals to reduce the ambiguity of the requirements employing patterns and linguistic rules; although part of our work is also to reduce the ambiguity of NL requirements, our work focuses on improving the requirements specification templates based on a standard template. (Arora, et al. 2013) and (Arora, et al. 2015) provide additional insight by supporting the automatic compliance of the requirements using NL processing techniques for the verification of requirements, something that our work does not raise and that will be part of later work on the automatic verification of the requirements. In addition, (Arora, et al. 2013) also presents a flexible template to specify requirements that can be adapted to different styles of writing requirements and other proposals such as (Souag, et al. 2018) go even further by allowing the automatic generation of non-functional requirements (security in particular) in semi-structured NL thanks to the use of two ontologies: a security ontology (Souag, et al. 2015) and a domain-specific ontology. In contrast, our proposal is agnostic to the type of non-functional requirements; however, it should be inspired in the future by the related work to facilitate the writing of requirements.

9 Conclusion and future works

The Rupp template has been established by the IREB as the *de facto* standard for the specification of individual requirements, however, when representing certain requirements at an industrial level this template is limited. For this reason we decided to study the research question related to (i) the gaps in the template used as a standard for writing requirements and (ii) how to fill those gaps. This study is based on the experimental research method called action research. This method allowed us to use consistently the authors' own experience in the field of requirements engineering, and the information available at the industrial level and in the literature (other templates and related works). As a result of this experiment, we identified the gaps in the Rupp template and, based on those gaps, we propose a more robust template that, unlike others, allows representing the quasi-totality of the requirements and constraints of two industrial cases.

Through this research we could observe that the reference template must be improved and that it is possible to improve it. We also found that the new template can be used in industrial cases; however, our research is still not conclusive considering that we only experiment with two industrial cases, however with the empirical evidence obtained so far we have

seen improvements in time and costs and the high-quality standards in most of the resulting requirements.

Some aspects that remain pending and require even more work are for example: restart the experiment starting from a different template than Rupp and compare the resulting template with that reported in this article; and implement a software tool for the automatic verification of requirements based on the improved template resulting from this research work. It is also necessary to study the improved template in other cases and other types of projects such as distributed, pervasive, cyber-physical, intelligent and data-intensive systems. Additionally, natural language patterns, standardized process verbs, and models that complement the improved template could be studied and used to complement this work.

In addition to the above, we aim at strengthening the empirical evidence regarding the advantages of the improved template, performing at least two other experiments in companies of different activity sectors to have more conclusive observations on the ease of use, completeness and the accuracy of the proposed template in other contexts. Another future work consists of complementing the template proposed in this article with the treatment of negative and ubiquitous requirements, among others.

Further rationale about the complexity introduced with the extensions are needed and will be addressed as part of the future work. In particular, we will study the following questions: how the extensions affect the usability and the comprehension of the templates obtained? And are the benefits obtained with the extensions defined significant concerning the complexity introduced?

References

Alexander, Ian, and Stevens Richard. Writing Better Requirements. Addison-Wesley, 2002.

Arora, Chetan, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Automated checking of conformance to requirements templates using natural language processing." IEEE Transactions on Software Engineering (Vol 41 No 10), 2015: 944-968.

Arora, Chetan, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. "Automatic Checking of Conformance to Requirement Boilerplates via Text Chunking: An Industrial Case Study." ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 2013: 35-44.

ARTEMIS. "Project CESAR." CESAR Partners. RSL Reference Manual. CESAR Consortium 1.1 Edition. 2010.

Baresi, Luciano, Liliana Pasquale, and Paola Spoletini. "Fuzzy Goals for Requirement-Driven Adaptation." 18th IEEE Int. Conf. Requirements Engineering (RE'10), 2010: 125-134.

Beck, Kent. "Embracing change with extreme programming." IEEE Computer. 32, 1999: 70–77.

Clegg, Dai, and Richard Barker. Case Method Fast-Track: A Rad Approach. Addison-Wesley, 1994.

Cohn, Mike. User Stories Applied for Agile Software Development. Adisson Wesley, 2004.

Cook, Thomas D, and D T Campbell. *Quasi-experimentation: Design & Analysis Issues for Field Setting.* Houghton Mifflin, 1979.

Davies, Rachel. *Format for expressing user stories.* 2001.

Denger, Christian, Daniel M. Berry, and Erik Kamsties. "Higher quality requirements specifications through natural language patterns." Proceedings of the IEEE International Conference on Software-Science Technology & Engineering: IEEE Computer Society, 2003: 80-90.

Ibrahim, Noraini, M.N. Wan Kadir, and Safaai Deris. "Documenting requirements specifications using natural language requirements boilerplates." 8th Malaysian Software Engineering Conference (MySEC), 20144: 19-24.

ISO/IEC/IEEE. "29148 Systems and software engineering (Life cycle processes — Requirements engineering)." 2011.

Jureta, Ivan J., Alexander Borgida, Neil A. Ernst, and John Mylopoulos. "The Requirements Problem for Adaptive Systems." ACM Trans. Management Inf. Syst. 5, 2015: 17:1-17:33.

Koshy, Elizabeth, Valsa Koshy, and Heather Waterman. Action research in healthcare. Sage, 2010.

Majumdar, Dipankar, Sabnam Sengupta, Ananya Kanjilal, and Swapan Bhattacharya. "Adv-EARS: A Formal Requirements Syntax for Derivation of Use Case Models." First International Conference on Advances in Computing and Information Technology, 2011: 40-48.

Majumdar, Dipankar, Sabnam Sengupta, Ananya Kanjilal, and Swapan Bhattacharya. "Automated Requirements Modeling with Adv-EARS." International Journal of Information Technology Convergence and Services, 2011: 57-67.

Mavin, A., P. Wilkinson, A. Harwood, and M. Novak. "Easy Approach to Requirements Syntax (EARS)." International Requirements Engineering Conference RE, 2009: 317-322.

Mazo, Raúl. Guía para la adopción industrial de líneas de productos de software. Medellín: Editorial Eafit, 2018.

Mazo Raúl. Software Product Lines, from Reuse to Self Adaptive Systems. Université Paris 1 Panthéon - Sorbonne, France, Habilitation à Diriger des Recherches (HDR), Octobre 2018.

Mazo, Raúl, and Carlos Jaramillo. "Hacia una nueva plantilla para la especificación de requisitos en lenguaje natural semi-estructurado." In the Proceedings of the Requirements Engineering Track (RET) of CIbSE. La Habana, Cuba, 2019.

Mazo, Raúl, Juan Muñoz-Fernández, Luisa Rincón, Camille Salinesi, and Gabriel Tamura. "VariaMos: an extensible tool for engineering (dynamic) product lines." XIX International Software Product Line Conference (SPLC), 2015: 374-379.

O'Brien, Rory. An Overview of the Methodological Approach of Action Research. In R. Richardson (Ed.), Theory and Practice of Action Research. Joao Pessoa: Universidade Federal da Paraíba, 2001.

Pohl, Klaus. Requirements Engineering - Fundamentals, Principles, and Techniques. Springer, 2010.

Robertson, Suzanne, and James Robertson. Mastering the Requirements Process: Getting Requirements Right, 3rd ed. Addison-Wesley, 2013.

Rupp, C. Requirements Engineering and Management. Hanser, 2007.

Sophist GmbH Requirements Templates - The Blueprint of your Requirement. SOPHIST GmbH. 2014. https://www.sophist.de.

Souag, Amina, Camille Salinesi, Raúl Mazo, and Isabelle Comyn-Wattiau. "A security ontology for security requirements elicitation." International Symposium on Engineering Secure Software and Systems (ESSoS), 2015: 157-177.

Souag, Amina, Raúl Mazo, Camille Salinesi, and Isabelle Comyn-Wattiau. "Using the AMAN-DA method to generate security requirements: a case study in the maritime domain." Requirements Engineering (Vol 23, Issue 4), 2018: 557–580.

Souza, V.E. Silva, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. "Awareness Requirements for Adaptive Systems." 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'11), 2011: 60-69.

Susman, Gerald I. Action Research: A Sociotechnical Systems Perspective. Sage, 1983.

Susman, Gerald I., and Roger D. Evered. "An Assessment of the Scientific Merits of Action Research." Administrative Science Quarterly (Vol. 23), 1978: 582-603.

Tjong, Sri Fatimah, Nasreddine Hallam, and Michael Hartley. "Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns." International Conference on Computer and Information Technology (CIT'06), 2006: 199-205.

Vassev, Emil. "Requirements Engineering for Self-Adaptive Systems with ARE and KnowLang." EAI Endorsed Transactions on Self-Adaptive Systems, 2015.

Whittle, Jon, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Brunel. "RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems." 17th IEEE International Requirements Engineering Conference (RE'09), 2009: 79-88.

Wiegers, Karl, and Joy Beatty. Software Requirements Third Edition. Microsoft Press, 2013.

Wieringa, Roel, and Ayse Morali. "Technical action research as a validation method in information systems design science." Design Science Research in Information Systems. Advances in Theory and Practice. DESRIST 2012. Lecture Notes in Computer Science. Springer, 2012. 220-238.