Taxonomy of Requirements Specification Templates

Hiba Hnaini, Raúl Mazo, Paola Vallejo, Jose Galindo, Joël Champeau

Taxonomy of Requirements Specification Templates

Hiba Hnaini*, Raúl Mazo*, Paola Vallejo†, Jose Galindo‡, and Joël Champeau*
*Lab-STICC, ENSTA Bretagne, Brest, France
{hiba.hnaini, raul.mazo, joel.champeau}@ensta-bretagne.fr
†GIDITIC, Universidad EAFIT, Medellín, Colombia
pvallej3@eafit.edu.co
‡Dpto. de Lenguajes y Sistemas Inf., Universidad de Sevilla, Sevilla, Spain
jagalindo@us.es

***Abstract*—Requirements specification is an early stage of system design. It consists of rephrasing and documenting stakeholders' explanations and needs in the form of clear and coherent requirements. However, these requirements are often expressed in natural language since it is the easiest communication method. Researchers have proposed semi-structured natural language templates or boilerplates for specifying functional and non-functional requirements, which consider security requirements. This seeks to enhance the quality of the requirements specifications and simplify their transformation to system models. However, it is still unknown what concepts, quality attributes, and good practices should be considered to specify requirements in a semi-structured natural language and how that information has been considered in the existing templates. In this paper, we aim to determine how templates are related among them and what are the implications (e.g., complexity, completeness, time) of using one or another. In this paper, we identify each template's concepts, quality attributes, and good practices by studying the template's aspects and then using a running example to formulate requirements using these templates. We also identify the aspects repeated or inherited from one template to another. This paper puts forward a taxonomy of requirements specification templates that categorize and specify the sources of the templates, which helps determine what template considers the aspects of another.**

***Index Terms*—Security Requirements Template, Boilerplate, Requirements Specification, Natural Language, Taxonomy.**

I. Introduction

The increase of security threats on software and hardware systems within the past few years has imposed consideration of security at all stages of system development, starting from the requirements specification stage. However, eliciting precise and non-complex security requirements can take time and effort. This challenge originates from the need for guides that help security requirements engineers define their requirements. Similarly to standard requirements, security requirements also aim that (i) the same interpretation is reached by all readers and (ii) this interpretation corresponds to the idea that the author of the requirement was trying to convey.

According to Denger *et al.* [1], the most common method for specifying and documenting requirements is Natural Language (NL) since it needs no training and is within reach. However, the drawbacks of NL are too significant to disregard. As Dalpiaz *et al.* [2] explain, these disadvantages are ambiguity, unclarity, inconsistency, and incompleteness. Mavin *et al.* [3] add the disadvantages of vagueness, complexity, duplication, verbosity, implementation, and untestability. Several researchers have proposed the use of semi-structured natural language in the form of guidelines, templates, boilerplates, patterns, and so on to mitigate some of the weaknesses of natural language when writing requirements. For example, a template controls the structure of the requirement by possibilities and restrictions while preserving the advantage of being in NL. In addition, this method reduces faults in the early stages of a system's development process.

Some proposals consist of generic guidelines that make it possible to specify requirements for almost any type of systems [4] [5].

Other proposals present different strategies to facilitate the work of the requirements engineers and, therefore, improve the quality of the products specified by modeling textual requirements. For example, through standard reference data [6], a generic syntactic requirements specification template [7], a robust template [8], or a template based on the 5W1H (Why, Who, Where, When, What, How) questions [9].

Other researchers defined and used templates for a specific domain. For example, Esser and Struss propose a natural language template-based interface to acquire requirements for functional testing of control software for passenger vehicles [10], and Mavin *et al.* present a set of structural rules to address common requirements problems, such as ambiguity, complexity, and vagueness [3]. The rule set allows all requirements to be expressed in natural language in one of five simple templates. The rule set was applied to extract requirements for an aero engine control system from an airworthiness standard document, and Mahmud *et al.* propose a toolchain for structured requirements specification in the Requirements Specification and Analysis (ReSA) language. "ReSA is an ontology-based requirements specification language tailored to automotive embedded systems development" [11].

Other researchers concentrated on security requirements specifications. For example, Toval *et al.* present a method for eliciting and specifying system and software requirements, including a repository containing reusable requirements, a spiral process model, and a set of requirements document templates [12]. Firesmith discussed the value of reusable parameterized templates for specifying security requirements [13]. Firesmith outlined an asset-based risk-driven analysis approach for determining the appropriate actual parameters to use when reusing such parameterized templates to specify se-

curity requirements that should improve the quality of security requirements in requirements specifications. Kamalrudin *et al.* propose a security requirements library and template to assist the requirements engineer in writing security requirements by providing them with the relevant sentence structure [14]. The library was built by compiling security attributes derived from parsing and keyword matching.

However, we could not find a knowledge resource that combines all the concepts, relations, and best practices needed during the requirements specification.

To fill the gaps identified in the state of the art, this paper proposes a taxonomy with the concepts and relationships of existing templates and guidelines for requirements specification. The taxonomy will give the requirements and constraints to create a new template.

The paper is organized as follows: The guidelines and templates used to form the taxonomy are presented and detailed in Section II. Section III briefly discusses the analyzed templates. Section IV presents a taxonomy of requirements specification templates. Finally, Section V concludes this work.

II. Baseline Guidelines and Templates

This section lists and explains the different templates and guidelines found in the literature for requirements specification. For a more precise illustration, we have used the example of the Blood Infusion Pump System presented by Lindvall *et al.* [15] to re-specify a requirement of the system using the guidelines and templates, where applicable.

A. Attempto Controlled English (ACE)

ACE, proposed by Schwitter and Fuchs [4], is a computer-processable subset of English with restricted grammar and domain-specific vocabulary that allows specialists to formulate requirement specifications. ACE is associated with its parser (APE) and a reasoner (RACE). ACE is structured by simple, composite, and interrogative sentences.

- **Simple sentences** express a true state of affairs, and their form is *subject* + *predicator* (+ *complement* + *adjunct*).
- **Composite sentences** are built from simpler sentences and constructors: coordination (*and*,*or*, *either-or*), subordination (*if-then*, *who/which/that*), and negation (*not*).
- **Interrogative sentences** allow users to pose *yes/no* questions and *wh*-questions.

ACE is based on syntactic, semantic, and disambiguation principles.

- **Syntactic principles** illustrate a form of the sentence of requirement. For instance, a syntactic principle is "nouns are always used with a determiner" (*e.g.*, "The customer enters a card").
- **Semantic principles** represent how different parts of the requirement are interpreted. For instance, "verbs denote events and states".
- **Disambiguation principles** are the steps taken to decrease and limit the ambiguity in the requirement. For instance, "some ambiguous constructs are not available" (*e.g.*, "John and Mary enter a card") and "unambiguous alternatives with scope markers can be used instead" (*e.g.*, "John and Mary enter a card each").

B. Gellish

Van Renssen presents an application-independent language that allows textual modeling requirements through standard reference data for system customization, data integration, and data exchange [6]. It is based on ontological concepts, where like in an ontology, the Gellish language represents the relationship between two objects. It adopts a table form to represent the language with the following columns:

- **Left-hand object UID:** a unique ID associated with the source object of the relationship.
- **Left-hand object name:** the name of the relationship's source object.
- **Fact UID:** a unique ID of the relationship (or fact).
- **Relationship type name:** the name of the relationship or fact. For example, *is a specialization of*, *is part of*.
- **Right-hand object UID:** a unique ID associated with the target object of the relationship.
- **Right-hand object name:** the name of the relationship's target object.

"Gellish is not limited to specific application domains, although the current ontology (the dictionary) does not yet cover the scope of a natural language". Table I illustrates a fact specified by Gellish.

TABLE I
Gellish Example.

Left-hand object UID	**Left-hand object name**	**Fact UID**	**Relationship type name**	**Right-hand object UID**	**Right-hand object name**
111	Andries	11	is classified as a	990007	man

C. Nayak et al.'s template

Nayak *et al.* propose a reliable requirements specification template [5] (based on Volere template [16]) with some parameters that evaluate the reliability of the individual requirement before finalizing the requirements documentation from the initial phase of software development. The template contains information about a requirement, such as Rationale, Source, Requirement Type, etc., but it focuses on the following reliability parameters: Severity Level, Confidence Level, and Rank of Requirement. However, this template does not give a structure for the requirement itself and considers it a description ("a one sentence statement of intention of the requirement") written entirely in natural language.

D. Rupp et. al.'s template

Rupp *et. al.* propose a generic syntactic requirements specification template, which focuses on the syntax (structure) of a requirement, not its semantics (content) [7]. The following parts compose the template.

- **Condition:** condition or constraint under which the requirement is to take place. *<When? Under what conditions?>*.
- **System:** name of the system concerned by the requirement. *THE SYSTEM <system name>*.
- **Priority:** degree of the legal obligation of the requirement. *SHALL, SHOULD, WILL, MAY*.
- **Functional activity:** functionality to be provided by the system:
  - Independent system action: *<process verb>*
  - User interaction: *PROVIDE <whom?> WITH THE ABILITY TO <process verb>*
  - Interface requirement: *BE ABLE TO <process verb>*
- **Object:** object concerned by the requirement. *<object>*.
- **Additional object details:** additional details or explanation about the object of the requirement, for example, *where?* or *how? <additional details about the object>*.

E. Easy Approach to Requirements Syntax (EARS)

Mavin *et al.* present a set of structural rules to address common requirements problems such as ambiguity, complexity, and vagueness [3]. It is based on ECA, "In ECA, the event specifies the signal that triggers the rule and the condition is a logical test that (if satisfied) causes the specified system action" [3]. The rule set allows all requirements to be expressed in natural language in one of the following simple templates.

- **Generic requirements syntax:** *<optional preconditions> <optional trigger> the <system name> shall <system response>*.
- **Ubiquitous requirements - no pre-condition:** *The <system name> shall <system response>*.
- **Event-driven requirements - triggering event:** *WHEN <optional preconditions> <trigger> the <system name> shall <system response>*.
- **Unwanted behaviors:** *IF <optional preconditions><trigger>, THEN the <system name> shall <system response>*.
- **State-driven requirements:** *WHILE <in a specific state> the <system name> shall <system response>*.
- **Optional features:** *WHERE <feature is included> the <system name> shall <system response>*.

F. Mazo et al.'s template

Mazo *et al.* identify some gaps in Rupp *et al.*'s template and, based on those gaps, propose a more robust template that facilitates the work of the requirements engineers and, therefore, improves the quality of the products specified with the new template [8]. Mazo *et al.*'s template is adapted to product lines and auto-adaptive systems (using the RELAX language presented in [17]). It has the following sections.

- **Conditions under which a behavior occurs:** describe behaviors performed or provided only under specific conditions (*e.g., IF, WHILE, IN CASE, AFTER*).
- **Family of systems, systems, or parts of a system:** allows specifying the name of the product line, system, subsystem, or system component (*e.g., ALL SYSTEMS OF THE <product line name>*).
- **Degree of priority:** specifies the degree of priority associated with a requirement (*i.e. SHALL, SHOULD, COULD, WILL*).
- **Activity:** specifies the characterization of the activity conducted by the system or the systems of a product line.
- **Object or objects:** specifies the object or objects that make up the system (*e.g., EACH <object>*).
- **Complementary details of the object(s):** can be one or more adjectives or a more enhanced description of the object.
- **Conditionality in the object:** describes a behavior that the system must execute if and only if the object attains the specified condition (*i.e., IF AND ONLY IF <condition>*.
- **Verification criterion (adjustment) of the requirement:** a detectable criterion to determine to what degree the requirement is verifiable.
- **Relax requirements statements for self-adaptive systems:** represents the autonomous nature of requirements in self-adaptive systems (*e.g., AS MANY, UNTIL*).

G. Cube

Pabuccu *et al.* present the Cube requirement writing template dedicated to software systems and based on 5W1H questions [9].

- **Why:** the goal of the requirement.
- **Who:** the actor of the requirement.
- **Where:** the place of the requirement.
- **When-trigger:** pre-condition of the requirement.
- **When-condition:** post-condition of the requirement.
- **What:** the requirement's action or activity.
- **How:** it explains how the system will be developed.

Cube identifies three types of requirements and provides three different templates:

- **Business Requirement:** *WHO - WHAT- WHERE - WHEN TRIGGER- WHEN CONDITION Aim, Reason: WHY*.
- **User Requirement:** - *when WHO - WHEN TRIGGER - WHERE - What - if - WHEN CONDITION - HOW (optional) - WHY (optional)*.
- **Functional Non-functional Requirement:** *Who - What - when - When Trigger -and When Condition - Where - WHY (optional) - HOW (optional)*.

H. Esser and Struss's template

Esser and Strauss propose a natural language template-based interface to acquire requirements for functional testing of control software for passenger vehicles [10]. The content of the filled-in templates can be represented in propositional logic and temporal relationships and form the correct expected behavior model. The template is structured as an ***if (start-condition) - then (consequence) - until (end-condition)*** sentence (*e.g.*, "if the system is in mode m1, lamp L3 is off,

and button B4 is released, then immediately lamp L3 is lit until button B4 is down again or the system leaves m1").

I. ReSA

Mahmud *et al.* propose a toolchain for structured requirements specification in the ReSA language [11]. "ReSA is an ontology-based requirements specification language tailored to automotive embedded systems development". ReSA "(i) renders natural language terms (words, phrases), and syntax, (ii) uses an ontology that defines concepts and syntactic rules of the specification, and (iii) uses requirements boiler-plates to structure specification". It is composed of six boilerplates or templates.

- **Simple:** statement that contains a modal verb, such as, *shall (e.g.*, "system shall be activated").
- **Proposition:** proposition or assertive statement *(e.g.*, "button is pressed".
- **Complex:** "is constructed from a Simple, Proposition boilerplate, and an adverbial conjunctive (such as *while, when, until*)", *e.g.*, "the error shall be reported while the fault is present".
- **Compound:** "is composed of two or more Simple or Proposition boilerplates and the logical operators, *AND/OR*" (*e.g.*, "system shall be activated and driver shall be notified").
- **Conditional:** "a different variant of conditional statements, *i.e.*, *if, if-else, if-elseif, or if-elseif-else*, and conditional nesting".
- **Prepositional Phrase:** "can be used to describe timing properties, the occurrence of events, and other complements to the subject of a main phrase" (*e.g.*, "within 5ms, by the driver".

J. SImple REuse of software requiremeNts (SIREN)

Toval *et al.* present a method for eliciting and specifying system and software requirements. The method includes a repository with reusable requirements, a spiral process model, and a set of requirements document templates [12]. The method focuses on information systems security. It does not provide a structured template for the requirements description but for the entire Software Requirements Specification document. The repository contains two types of requirements: **parameterised** (some parts have to be adapted to the application being developed at the time, *e.g.*, "The security manager shall check the user's identifiers every [time in months] to detect which ones have not been used in the last [time in months]") and **non-parameterised** (could be applied directly to any project concerning the repository's profiles and/or domains, *e.g.*, "The firewall configuration will be screened host").

K. Firesmith template

Firesmith discusses the value of reusable parameterized templates for specifying security requirements [13]. Firesmith outlined an asset-based risk-driven analysis approach for determining the appropriate current parameters to use when reusing parameterized templates to specify security requirements that should improve the quality of security requirements in requirements specifications. Firesmith explains how to create templates and gives an example of a security requirement to specify integrity: "The [application/component/data center/business unit] shall protect the [identifier—type] data it transmits from corruption (*e.g.*, unauthorized addition, modification, deletion, or replay) due to [unsophisticated/somewhat sophisticated/sophisticated] attack during execution of [a set of interactions/use cases] as indicated in [specified table]". However, no general template was provided.

L. Kamalrudin et al.'s template

Kamalrudin *et al.* propose a security requirements library and template to assist the requirements engineer in writing security requirements by providing them with the sentence structure [14]. The library was built by compiling security attributes derived from parsing and keyword matching. The template provided is as follows: *The <Subject> should <Verb/Security Keyword> to the <Object> <Security Keyword> <Security Mechanism> in order to <Adjective Phrase>* (*e.g.*, "[The] customer should register [to] the system using unique username and password in order to proceed to book ticket service").

M. AMAN-DA

Souag *et al.* propose a security requirements elicitation and an analysis method [18]. It uses a multi-level domain ontology of security notions and provides a security requirements template: *<When><Under what condition> <Agent name> "Shall/Should/Will" <Action> <Assets> <Additional Information>* (*e.g.*, "The web publishing system should lock accounts after reaching logon failure threshold").

III. Application Example

Table II shows the re-specification of some Blood Infusion Pump System requirements using templates discussed in Section II. Per the result of the requirements re-specification and the information provided in Section II concerning each guideline and template, we were able to categorize them according to their structure or nature, the type of requirements they can represent, and their output. We also noticed a similarity in the structure of some templates. It is because they were based on a combination of other ones. For example, Mazo *et al.*'s template uses the structure of Rupp *et al.*'s and the EARS templates. This categorization resulted in the creation of the taxonomy presented in Section IV. For example, it is explained by Mavin *et al.*, and clear in the structure of EARS, that the EARS template integrates the event-condition-action concepts of the Event-Condition-Action (ECA) language or template. This is also true in the Mazo *et al.* template, which inherits some concepts, quality attributes, and good practices from the Rupp *et al.* and EARS templates.

IV. Taxonomy

To identify the source and the guidelines and templates used in constructing each template, we propose a taxonomy of all

TABLE II
EXAMPLE REQUIREMENTS USING THE TEMPLATES.

Template	Example Requirement - Blood Infusion Pump System
ACE	**R1.2** If the silent warning alarm persists for 10 seconds or more, then the system shall trigger an audible warning alarm
	R1.2.ACE If the silent warning alarm persists for 10 seconds or more, then the system triggers an audible warning alarm
Rupp *et al.*	**R2.5:** If BP improves from true critical condition to true warning condition, then the system shall trigger a true warning alarm
	R2.5.Rupp: <If BP improves from true critical condition to true warning condition then>$_{condition}$<the blood infusion pump system>$_{system}$ <shall>$_{priority}$ <trigger>$_{activity}$ <an audible critical alarm>$_{object}$
EARS	**R2.8:** The system shall disable the "Start infusion" button whenever blood infusion is stopped due to critical alarm
	R2.8.EARS: The <blood infusion pump system>$_{systemname}$ shall <disable the "Start infusion" button whenever blood infusion is stopped due to critical alarm>$_{systemresponse}$
Mazo *et al.*	**R3.1:** If the system received two consecutive out-of-range BP low values (less than 10) or high values (greater than 180), then the system shall initiate system shutdown
	R3.1.Mazo: If <the system received two consecutive out-of-range BP low values (less than 10) or high values (greater than 180)>$_{condition}$ then <the blood infusion pump system>$_{system}$ <shall>$_{priority}$ <initiate>$_{activity}$ <system shutdown>$_{object}$
Cube	**R1.6:** The system shall disable the 'Start Infusion' button during the warning alarm
	R1.6.Cube: <The system>$_{who}$ <shall disable the 'Start Infusion' button>$_{what}$ <when the warning alarm starts>$_{whentrigger}$<and enable the button after the alarm stops>$_{whencondition}$
Esser and Struss	**R2.2:** If the silent critical alarm clears within 10 seconds and there was a true warning alarm before, then the system shall resume blood infusion and clear the warning alarm
	R2.2.Esser: If -the silent critical alarm clears within 10 seconds and there was a true warning alarm before-, then -the system shall resume blood infusion and clear the warning alarm- until ?
ReSA	**R1.3:**If BP improves, then the system shall resume blood infusion and clear the warning alarm
	R1.3.ReSA: <If BP improves,>$_{conditional}$ <the system shall resume blood infusion and clear the warning alarm>$_{compound}$
Kamalrudin *et al.*	**S2.2:** The system shall use the authenticators to validate the source of the BP value
	S2.2.Kamal: <The system>$_{subject}$ <shall authenticate>$_{verb}$ <the BP value>$_{object}$ <using>$_{Security-keyword}$ <an authenticator>$_{security-mechanism}$ <to validate the source of the BP value>$_{adjectivephrase}$
AMAN-DA	**S4.1.1:** A known-good cryptographic algorithm shall be used to implement authentication.
	S4.1.1.AMAN-DA: <The system>$_{agent}$ <shall><implement >$_{action}$ < authentication>$_{asset}$ <using a known-good cryptographic algorithm >$_{additionalinformation}$

the above templates in Figure 1. A relationship between two templates means that the target (target of the arrow) template has been considered or studied when constructing the source template (source of the arrow). The benefit of having such a taxonomy is to avoid repeating already included templates when building new requirement templates. The taxonomy categorizes the templates and guidelines between controlled-natural languages and templates. Then, each category determines the type of requirements that can be represented by each template (functional, non-functional, or security requirements). There are also two types of templates identified in the taxonomy, templates for requirements description and templates for System Requirements Specification (SRS) documents, as shown in the legend of Figure 1. For example, as we have mentioned in Section III, Mazo *et al.*'s template integrates some aspects of the Rupp *et al.* and EARS templates. It is evident through the example given in Table II that the syntax of the requirement formulated with the Mazo *et al.*'s template is similar to the requirement reformulated with the Rupp *et al.*'s template. Both requirements are composed of a condition, a system name, a priority keyword, an action, and an object. This similarity is described in Figure 1 by the relationship between Mazo *et al.*'s template (source) and the EARS and Rupp *et al.* templates (targets). This relationship indicates that Mazo *et al.*'s template already respects and integrates the concepts, quality attributes, and good practices from Rupp *et al.* and EARS. Thus, it is unnecessary to reconsider them when using the Mazo*et al.*'s template, which reduces the time and resource cost.

V. CONCLUSION

System security has become a very critical issue to handle at all stages of system design, starting from the requirements specification stage. To guide and simplify this stage, several requirements templates have been proposed. We have discussed the concepts, attributes, and good practices and examined these templates in Section II. Then, in Section III, we re-specified some requirements of the Blood Infusion Pump System using the templates found. We could categorize the templates and guidelines according to different characteristics in this process. However, most of these templates were built based on other templates, where they integrate the concepts, quality attributes, and good practices of others. With the categorization of these templates and the identification of the sources of each template, we were able to create a taxonomy. The taxonomy presented in Section IV aims to avoid the repeated study of source templates which is a loss of research hours and resources. We intend to use this taxonomy as a baseline to create a new security requirements specification template where we can identify, using the taxonomy, the templates we can base our new template on while avoiding re-studying already integrated templates.

ACKNOWLEDGMENT

We thank the French DGA (Direction Générale de l'Armement), the European Union, and the Spanish Government for supporting this research. This work was supported by the project *Data-pl* funded by FEDER/Ministry of Science and Innovation — State Research Agency; the *COPERNICA*

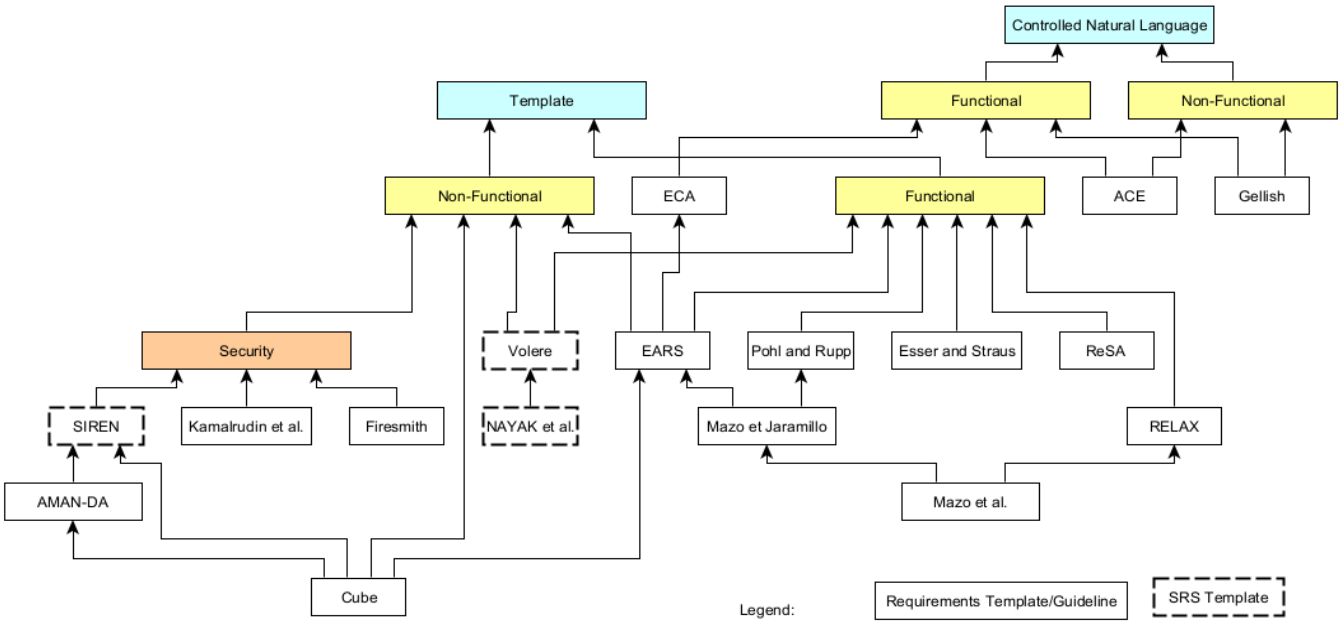

Fig. 1. Requirements templates and guidelines taxonomy.

(P20_01224) and *METAMORFOSIS* (FEDER_US-1381375) projects funded by Junta de Andalucía.

REFERENCES

[1] C. Denger, D. M. Berry, and E. Kamsties, "Higher quality requirements specifications through natural language patterns," *Proceedings 2003 Symposium on Security and Privacy*, pp. 80–90, 2003.

[2] F. Dalpiaz, I. Schalk, and G. Lucassen, *Pinpointing Ambiguity and Incompleteness in Requirements Engineering via Information Visualization and NLP*, 03 2018, pp. 119–135.

[3] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *Proceedings of Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*. United States: IEEE, Nov. 2009, pp. 317–322, 2009 17th IEEE International Requirements Engineering Conference ; Conference date: 31-08-2009 Through 04-09-2009.

[4] N. Fuchs and R. Schwitter, "Attempto controlled english (ace)," *CoRR*, vol. cmp-lg/9603003, 03 1996.

[5] S. K. Nayak, R. A. Khan, and M. R. Beg, "A comparative template for reliable requirement specification," *International Journal of Computer Applications*, vol. 14, no. 2, pp. 27–30, 2011.

[6] A. van Renssen, "Gellish: an information representation language, knowledge base and ontology," *ESSDERC 2003. Proceedings of the 33rd European Solid-State Device Research - ESSDERC '03 (IEEE Cat. No. 03EX704)*, pp. 215–228, 2003.

[7] C. Rupp, M. Simon, and F. Hocker, "Requirements engineering und management," *HMD Praxis der Wirtschaftsinformatik*, vol. 46, pp. 94–103, 06 2014.

[8] R. Mazo, C. M. Z. Jaramillo, P. Vallejo, and J. M. Medina, "Towards a new template for the specification of requirements in semi-structured natural language," *J. Softw. Eng. Res. Dev.*, vol. 8, p. 3, 2020.

[9] Y. U. Pabuccu, I. Yel, A. B. Helvacioglu, and B. N. Asa, "The requirement cube: A requirement template for business, user, and functional requirements with 5w1h approach," *International Journal of Information System Modeling and Design (IJISMD)*, vol. 13, no. 1, pp. 1–18, 2022.

[10] M. Esser and P. Struss, "Obtaining models for test generation from natural-language-like functional specifications," pp. 75–82, 2007.

[11] N. Mahmud, C. Seceleanu, and O. Ljungkrantz, "Resa tool: Structured requirements specification and sat-based consistency-checking," in *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2016, pp. 1737–1746.

[12] A. Toval, J. Nicolás, B. Moros, and F. García, "Requirements reuse for improving information systems security: a practitioner's approach," *Requirements Engineering*, vol. 6, no. 4, pp. 205–219, 2002.

[13] D. Firesmith, "Specifying reusable security requirements." *J. Object Technol.*, vol. 3, no. 1, pp. 61–75, 2004.

[14] M. Kamalrudin, N. Mustafa, and S. Sidek, "A template for writing security requirements," in *Asia Pacific Requirements Engeneering Conference*. Springer, 2017, pp. 73–86.

[15] M. Lindvall, M. Diep, M. Klein, P. Jones, Y. Zhang, and E. Vasserman, "Safety-focused security requirements elicitation for medical device software," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 134–143.

[16] J. Robertson and S. Robertson, "Volere," *Requirements Specification Templates*, 2000.

[17] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in *2009 17th IEEE International Requirements Engineering Conference*, 2009, pp. 79–88.

[18] A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Using the aman-da method to generate security requirements: a case study in the maritime domain," *Requirements Engineering*, vol. 23, no. 4, pp. 557–580, 2018.