

Article

SURE: Structure for Unambiguous Requirement Expression in Natural Language

Franklin Parrales-Bravo ^{1,2,*}, Rosangela Caicedo-Quiroz ³, Julio Barzola-Monteses ^{1,4,*},
Leonel Vasquez-Cevallos ⁵, María Isabel Galarza-Soledispa ² and Manuel Reyes-Wagnio ²

- ¹ Grupo de Investigación en Inteligencia Artificial, Facultad de Ciencias Matemáticas y Físicas, Universidad de Guayaquil, Guayaquil 090514, Ecuador
 - ² Carrera de Ingeniería de Software, Facultad de Ciencias Matemáticas y Físicas, Universidad de Guayaquil, Guayaquil 090313, Ecuador; maria.galarzas@ug.edu.ec (M.I.G.-S.); manuel.reyesw@ug.edu.ec (M.R.-W.)
 - ³ Centro de Estudios de Didáctica y Gestión de la Educación Superior, Universidad Bolivariana del Ecuador, km 5 ½ vía Durán—Yaguachi, Durán 092405, Ecuador; rcaicedoq@ube.edu.ec
 - ⁴ Centro de Estudios en Tecnologías Aplicadas, Universidad Bolivariana del Ecuador, km 5 ½ vía Durán—Yaguachi, Durán 092405, Ecuador
 - ⁵ Simulation Clinic SIMUEES, Universidad Espíritu Santo, Samborondón 092301, Ecuador; leonelasvasquez@uees.edu.ec
- * Correspondence: franklin.parralesb@ug.edu.ec (F.P.-B.); jjbarzolam@ube.edu.ec (J.B.-M.)

Abstract: This study presents three structures for clearly expressing functional requirements (FRs) and quantitative non-functional requirements (qt-NFRs). Expressing requirements with these structures will allow the understanding of requirements by stakeholders and software developers. The first structure is the SURE format, which is composed of three main sections: a title, a short definition, and a detailed description. The second proposed structure is a template to facilitate the definition of the title and description of unambiguous FRs. It is based on the application of CRUD operations on a certain entity, calling it the “CRUDE” structure. Finally, the third structure serves as a template to make it easier to clearly define the description and title of qt-NFRs. It is based on the application of system properties to computer events or actions, calling it the “PROSE” structure. In this, it is very important to specify those metric values that are desired or expected by the stakeholder. To know how much the definition of FRs and qt-NFRs improved when the proposed structures were used, 46 requirement specification documents elaborated as homework by students of the “Requirement Engineering” course offered at the University of Guayaquil between 2020 and 2022 were evaluated by five experts with more than 10 years of experience in software development for Ecuadorian companies. The findings showed that students reduced the percentage of unambiguous FRs and qt-NFRs from over 80% to about 10%. In conclusion, the findings demonstrate how crucial the three structures proposed in this paper are to helping students develop the ability to clearly express requirements.

Keywords: requirement specification; functional requirements; non-functional requirements; requirements engineering education; software engineering



Citation: Parrales-Bravo, F.; Caicedo-Quiroz, B.; Barzola-Monteses, J.; Vasquez-Cevallos, L.; Galarza-Soledispa, M.I.; Reyes-Wagnio, M. SURE: Structure for Unambiguous Requirement Expression in Natural Language. *Electronics* **2024**, *13*, 2206. <https://doi.org/10.3390/electronics13112206>

Academic Editor: Arkaitz Zubiaga

Received: 8 May 2024

Revised: 21 May 2024

Accepted: 23 May 2024

Published: 5 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The four steps of requirements engineering (RE) that are usually included in it are the elicitation, analysis, validation, and specification of requirements [1]. This manuscript will focus on the last step. According to [2], the full expression (specification) of a requirement is not just a simple statement of stakeholder needs; it must be expressed in language that everyone involved can clearly understand. It implies that the requirement must be understandable to:

1. Those in charge of putting the requirement into practice (software developers).
2. Those in charge of demonstrating that the built system satisfies the requirement.
3. Those in charge of demonstrating that the system satisfies the needs of the stakeholder.

Thus, a requirement specification may be expressed as follows: A requirement statement without technical words (user requirement) and the technical characteristics of the requirement, useful for the development team (system requirement) [3]. Furthermore, requirements are classified into several categories, among which functional (FRs) and non-functional (NFRs) requirements stand out [1,4].

One of the main issues with large-scale projects is the lack of clarity when expressing software requirements [5]. Uncertain requirements may result in the production of unsatisfactory software systems [6]. Therefore, convincing the stakeholder to define their product requirements is one of the most crucial success factors [5,6].

FRs and NFRs are frequently specified using Natural Language (NL) [7]. Nonetheless, all specifications in NL have inconsistent and underspecified terms [8]. The Spanish [9,10] and English [6] languages have contradictory meanings and ambiguous syntax. In addition, the NFRs are not clear, are poorly recorded, are communicated informally, and are poorly understood [11]. Furthermore, FRs and NFRs across domains are presented quite differently depending on the company culture and organizational structure [12,13]. As a result, the RE community has been researching the common issue of ambiguity when expressing requirements in NL for about 20 years [13].

Templates have proven to be an effective tool for reducing ambiguity when requirements are expressed in NL [14]. However, to the best of our knowledge, there is no single syntactic structure that is ideal for specifying unambiguous FRs and NFRs in NL. Rather, recommendations for requirements quality, professional judgment, industry standards, and best practices are used to express clear requirements [15,16]. All of this motivates us to develop frameworks that help students define clear user and system requirements.

The major contributions of the manuscript are as follows:

- For expressing understandable requirements for both stakeholders and software developers, a format called SURE is proposed, which is composed of three main sections: a title, a short definition, and a detailed description.
- To facilitate the definition of unambiguous FRs in NL by students, the CRUDE structure is proposed, which is based on applying CRUD operations to a particular entity.
- To help students define unambiguous qt-NFRs in NL, the PROSE structure is presented. It is based on applying characteristics to a computer system or an action/event thereof. This manuscript provides a table with system properties and suggested metrics to facilitate the specification of those metric values that the stakeholder wants/expects.
- The CRUDE and PROSE structures have been tested on 46 requirements specification documents requested as homework in RE courses taught at the University of Guayaquil between 2020 and 2022. The results demonstrated that students decreased the percentage of unambiguous FRs and qt-NFRs from more than 80% to around 10%. The results show, in summary, how important those structures are to the student's capacity to express requirements clearly and concisely.

The remaining sections of the paper are structured as follows: In Section 2, prior research relevant to this investigation is provided. Section 3 offers a concise overview of the proposed structures to unambiguous requirement specifications in NL. The results and discussion of the outcomes are detailed in Section 4. Lastly, Section 5 encompasses concluding marks.

2. Related Work

2.1. On Functional and Non-Functional Requirements Definitions

Regarding what constitutes functional requirements (FRs), there is a very wide agreement. It expresses a function that a system must be able to perform and what the system ought to do [4]. In other words, FRs are defined as those requirements that specify the inputs to the system, the outputs (responses) from the system, and behavioral relationships between them [1].

On the other hand, non-functional requirements (NFRs) are important for the success of a product [11]. However, within the requirements engineering community, there is

currently no agreement on what NFRs are or how best to elicit, express, and validate them [11,17]. Moreover, NFRs are inadequately documented, conveyed informally, poorly understood, and unclear [11].

Among the most accepted classifications of NFRs is dividing them between quantitative and qualitative [18]. Compliance with quantitative NFRs (qt-NFRs) is verified with some quantitative measurement [19], while qualitative NFRs (ql-NFRs) do not have direct verification. In the latter, compliance can be verified through a subjective judgment of the interested parties about the implemented system through prototypes [17]. In this study, we will focus on the first ones (qt-NFRs), while the ql-NFRs will be the object of study in future work due to their complexity.

2.2. Specification Issues in Natural Language

While the format for requirements descriptions are still being explored, it is important to choose a language that both developers and stakeholders can readily comprehend, as stated in [20]. The authors also state that there are many ways to describe requirements: NL, goal-oriented, scenario-based, formal language, UML, logical description, and more. Between them, NL has been around for a long time and is even recommended as the best language for requirement specification [7]. Furthermore, NL enables diverse stakeholders—who can possess differing experiences and backgrounds—to exchange knowledge with one another [13].

Beyond the advantages, requirements expressed in NL often present ambiguous terms as well as unequal use of language [8]. For example, in 40 approved NASA requirements specification documents, some problems were identified in the language used to convey requirements and the structure of individual requirement statements [21]. To overcome those issues, the work presented in [3] suggests the use of requirements templates and patterns for information systems that might help in expressing requirements during the requirements elicitation phase of RE, maintaining the benefits of using NL, and avoiding an early formalization of requirements. Moreover, controlled NL representations for expressing software requirements have been proposed by [22–25], these will be discussed in detail below.

2.3. Templates for Functional Requirements

Some templates have been raised for facilitating the definition of FRs. However, beyond advantages, they have some limitations that we will mention in the following lines:

- In [3], some templates and patterns have been proposed and taken into consideration for several scholarly projects; actual software firms in Spain have even successfully employed them. Nevertheless, some characteristics of the information gathered in the “Specific Data” section of their proposed formats are not taken into account in their study. For instance, it is not stated what type of data each attribute is or how long it is. Additionally, there are no guidelines provided to prevent the creation of unclear requirements in the “Descriptive Name” portion of their proposed format, and amalgamations of requirements such as “Sales Module” or “Register Data” can be defined.
- The book presented by [22] proposes a template that divides a requirement statement’s syntactic structure into many pre-defined slots. Six slots are presented in their template, namely: (1) an optional precondition; (2) the name of the system; (3) a modal (shall/should/will) indicating the importance of the requirement; (4) the necessary processing functionality (this slot can accept three forms depending on how the functionality is to be rendered); (5) the object for which the functionality is required; and (6) optional extra details about the object. However, some requirements are excluded from that template, such as “For each communication channel type, the system needs to maintain a configurable timeout parameter” [14]. Moreover, when the glossary terms are unknown, this method is ineffective because it is unable to identify sentence fragments that match the template slots [14].

- In [23], a template called “Easy Approach to Requirements Syntax” (EARS) is proposed. In it, four slots are considered, namely, (1) an optional precondition; (2) the system name; (3) a modal; and (4) the system response that illustrates the system’s behavior. However, it only describes user FRs, omitting the details of attributes, data types, and data format, details that are valuable to software developers.

All in all, some templates have been proposed to facilitate the definition of FRs. However, they have some opportunities for improvement, such as (1) the lack of guidance to avoid creating ambiguous requirements, (2) the existence of FRs that do not fit the template, or (3) the omission of details of attributes, data types, and data format, details that are valuable to software developers.

2.4. Templates for Non-Functional Requirements

A few templates have been proposed to make the process of defining qt-NFRs easier. Besides benefits, they do have certain drawbacks, which we shall discuss, as follows:

- In [24], the “SENoR” method for eliciting NFRs is presented. In it, the use of Non-functional Requirements Templates (NoRTs) for defining qt-NFRs are proposed. It consists of three parts, namely,
 1. Core-basis: this refers to the crucial, unchanging portion of a statement.
 2. Parameters: this refers to a collection of variables, often an object’s attributes, the value of which needs to be ascertained in order to make an NFR specific and measurable.
 3. Optional parts: this refers to a group of selectable sentences. The amount of phrases that can occur in an NFR depends on the type of optional part, i.e., (1) 0..*—zero or more; and (2) 1..*—one of more.

Beyond the valuable quantification of NFRs proposed in NoRTs, it only defines a requirement at the user level (user requirement) and does not describe the technical aspects that allow compliance with it (system requirement). Furthermore, for inexperienced requirements elicitors, learning the NoRT structure to define qt-NFRs can be a bit complex since the manuscript does not provide any lists of parameters or optional parts to choose from.

- In [25], the use of a list of ISO25010 subcharacteristics is proposed for the definition of qt-NFRs through the NoRTs template. However, a list of metrics to define the value expected by the stakeholder in each sub characteristic is not provided, making its use difficult for inexperienced requirements elicitors.

All in all, some templates have been proposed to facilitate the definition of qt-NFRs. However, they have some opportunities for improvement, such as (1) the non-specification of the technical details to comply with what was requested by the stakeholder (system requirement), and (2) the absence of a list of metrics that facilitates the specification of values expected by the stakeholder in each qt-NFR.

2.5. Our Proposal

In this work, we may utilize parts of the format described by [3], but we are adding some structures to prevent students in the University of Guayaquil’s Software degree program from unambiguously defining FRs and qt-NFRs at both the user and system levels.

For specifying FRs, we will relax the definition of user FRs provided by [22,23] being able to use the same ones (or not) but emphasizing the application of a CRUD operation on an entity. In addition, we address the definition of the system’s FR with the subsequent definition of entity attributes, as well as the data types, length, and format restrictions (if necessary) for each attribute that is involved in the requirement.

Additionally, for facilitating the definition of qt-NFRs, we will relax the NoRT template proposed by [24,25], using the same idea, which is the definition of the expected value of a particular property of the system. Finally, to facilitate the definition of qt-NFRs by

inexperienced requirements elicitors, we will provide a list of computer system properties and metrics.

3. Proposed Structures

The three proposed structures for unambiguous requirement expression in NL are described in detail in this section.

3.1. Format SURE to Specify User and System Requirements

To specify user and system requirements, we propose the Structure for Unambiguous Requirement Expression (SURE), which is shown in Figure 1. Three primary sections are contemplated in it, namely: a title, a short definition, and a detailed description of each FR or qt-NFR. The short definition will contain the specification of a user requirement, and the details will include the definition of a system requirement. Both may be distinguished thanks to the SURE format, allowing the understanding of requirements by stakeholders and software developers. As mentioned (Section 2.5), our SURE format is based on the one described by [3].

CODE:	
TITLE:	User requirement
SHORT DEFINITION:	
DESCRIPTION AND RESTRICTIONS DETAILS:	System requirement
REVISION DATE AND VERSION:	
PRIORITY:	

Figure 1. Structure of the SURE format, used to specify both user and system requirements in software projects.

When defining requirements using the SURE format, steps shown in Figure 2 must be adhered to.

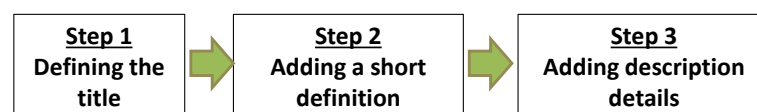


Figure 2. Steps to specify both user and system requirements in software projects through the SURE format.

Templates proposed for expressing FRs and qt-NFRs, such as EARS [23] or that presented by [22] for the case of FRs and NoRTs [24,25] for the case of qt-NFRs, can be used within step 2 to describe user requirements. However, we have proposed two structures to facilitate the unambiguous definition of both FRs and qt-NFRs. Below are specifics on the structures and how the three main sections of the SURE format should be filled to define FRs and qt-NFRs.

3.2. Using the SURE Format to Specify Functional Requirements

An example of FR specification in the SURE format can be seen in Figure 3. The requirement “Register soccer referees” for a system that manages the assignment of referees and the information generated by them in each match of a certain soccer sports championship is described as an example of a FR and illustrated in Figure 3.

It is crucial to note that there are other ways to describe FRs outside of using the SURE format. For example, while outlining interface requirements, it is vital to consider the use of low-fidelity or high-fidelity prototype tools, including Figma and Adobe Xd. In any case, the use of the SURE format is recommended for those cases in which CRUD operations applied to an entity are addressed.

CODE:	RF-1	User requirement
TITLE:	Register soccer referees	
SHORT DEFINITION:	The system must allow the registration of the following fields: Name, Last name, Age, Training received, Sex, Membership status, Email, User	
DESCRIPTION AND RESTRICTIONS DETAILS:	The system must allow the registration of the following information: • Name. It must contain a maximum of 32 characters, including uppercase and lowercase letters. • Last name. It must contain a maximum of 28 characters, including uppercase and lowercase letters. • Age. The age of the referees will only be written in numbers. • Years of experience. It should be written only in numbers. • Training received. It will not have a character limit, it will consider uppercase, lowercase, numbers and symbols. • Sex. It will indicate if the sex is female or male • Membership status: enabled or not enabled • Email. It must contain a maximum of 32 characters, including numbers and symbols. • User. It must contain a maximum of 12 characters including lowercase, uppercase, numbers and symbols.	
REVISION DATE AND VERSION:	26/12/2020 Version1.0	System requirement
PRIORITY:	High	

Figure 3. Example of a FR expressed in the SURE format.

Below is a detailed explanation of how to fill the title, brief definition, and description sections when expressing FRs in the SURE format.

3.2.1. Using the CRUDE Structure to Define the Title

The structure proposed to construct the titles of FRs is based on CRUD (Create, Read, Update, and Delete) database operations. We apply these CRUD activities on an entity to express FR. We have called it the CRUDE structure (CRUD operator + Entity).

The following are examples of CRUD actions applied on the “User” entity: Create, Read, Update, and Delete Users. Synonyms for CRUD operations also enable the definition of FR’s titles while adhering to the same structure, i.e., CRUD operator + Entity. Table 1 displays a selection of titles that follow the CRUDE structure.

Table 1. CRUD synonyms.

CRUD Operator	Synonyms
Create	register, add, insert
Read	search, list, generate report, print report
Update	edit, change, modify, update, correct
Delete	remove, drop, omit, cancel

FRs that follow the recommended CRUDE structure avoid merging many requirements into one. Determining the title of a FR as a “customer module” is an example of combining many requirements. The word “module” embodies a combination of several operations,

since this module is responsible for managing the registration, modification, search, and deletion of customer records.

Although the term “module” does not explicitly relate to any kind of object or entity, entities might utilize this word to group FRs related to any particular entity. For example, the FRs for the customer module could include “register customer”, “search customer”, “modify customer”, and “delete customer”. Furthermore, this module may also have the following FRs: “list customers by ID” and “generate customer report by date”.

Since “data” does not relate to a specific entity but to its records, another example of ambiguity is defining “register data” as the title of any FR. It is thus recommended that before enrolling in the RE course, students have some acquaintance with object-oriented modeling.

3.2.2. Adding a Short Definition

The SURE format includes a section called “short definition” that provides a synopsis of the properties that the system is required to manage for a particular entity. As shown in the example of Figure 3, the data usually relate to the fields that require updating, recording, or those considered as search fields. More examples of short definitions of FRs are shown in Table 2.

Table 2. Examples of titles and short definitions of FRs.

Title	Short Definition
Register enrollment	The system will request the registration of the following information: The ID of enrollment, student code, course code, academic cycle, status, and timestamp. The status will be equal to ‘A’ (active). A timestamp will be added by the system.
Cancel enrollment	The system must allow the cancellation of registration associated with a student and a course. To this, the status will be changed from ‘A’ (active) to ‘I’ (inactive).
Make payment	The system will request the registration of the following information: ID, invoice code that you wish to pay, and the user code that processed the payment. A timestamp will be added by the system.
Register course	The system will request the registration of the following information: course code, title, description, time interval, and registration limit (quota). A timestamp will be added by the system.
Search course by title	The system must allow the search of courses by title.
Register soccer referees	The following information must be collected and registered by the system: Name, Last name, Age, Training received, Sex, Membership status, Email, and User.

You can additionally specify the type(s) or profiles of people that can access to these operations. Stated differently, the “description” section contains technical details like attribute length and data type that support the user’s FRs, which are described here.

3.2.3. Adding Description Details

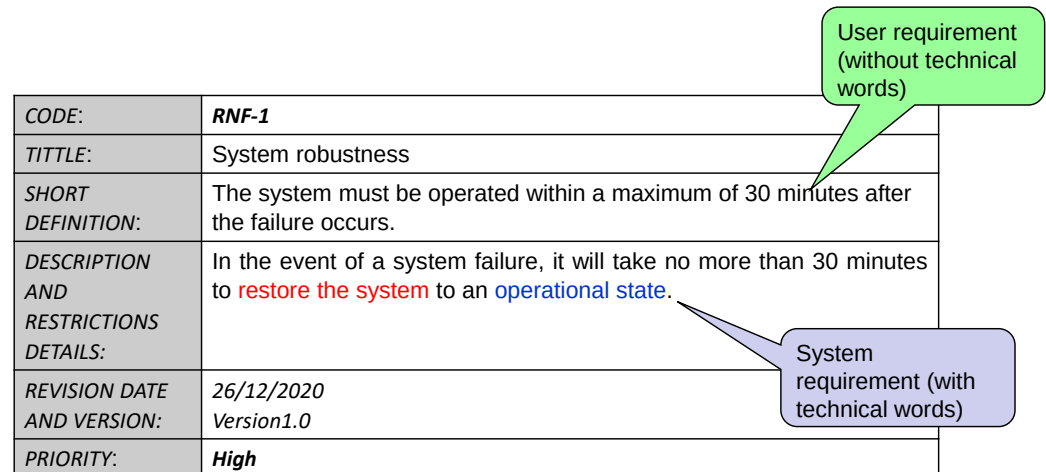
The “Description and constraints details” section of the SURE format should describe information useful to software developers, such as the length and data type of each attribute, in addition to the brief explanations provided in the preceding section. For instance, Table 2’s brief description of the title requirement “Register soccer referees” includes information on each of the characteristics, which is displayed in detail in Figure 3. In some cases, it is necessary to provide the format in which the data will be represented. An example of this would be “MAT00001”, which is a registration code consisting of five integers after the prefix “MAT”.

In addition, some sequence of steps must be indicated so that the user can perform the required operation. However, more details of them can be better modeled using some

behavioral diagrams such as sequence diagrams, time diagrams, collaboration diagrams, automation, data flow diagrams, etc.

3.3. Using the SURE Format to Specify Non-Functional Requirements

An example of qt-NFR specification in the SURE format can be seen in Figure 4. The short definition contains the specification of a user qt-NFR and the description contains the definition of a system qt-NFR. The requirement “System robustness” is presented as an example of a qt-NFR and is illustrated in Figure 4.



CODE:	RNF-1
TITLE:	System robustness
SHORT DEFINITION:	The system must be operated within a maximum of 30 minutes after the failure occurs.
DESCRIPTION AND RESTRICTIONS DETAILS:	In the event of a system failure, it will take no more than 30 minutes to restore the system to an operational state .
REVISION DATE AND VERSION:	26/12/2020 Version1.0
PRIORITY:	High

Figure 4. Example of a qt-NFR expressed in the SURE format. Technical words are marked with blue and red text.

3.3.1. Using the PROSE Structure to Define the Title

The structure proposed to build the titles of qt-NFRs is based on the application of system properties to a particular system or event. It will be called the PROSE structure (Properties + System/Event). Table 3 presents properties of a software system as well as suggested metrics that allow specifying the values expected by the stakeholders to consider the requirement fulfilled.

The criteria used to select the metrics suggested in Table 3 consisted of reviewing those that have been proposed in the literature. They are cited in the “Reference” column of the table. We have not limited the search to those works published during the last 5 years, since relevant software metrics to evaluate the properties of computer systems have been proposed long before.

To validate the suggested metrics of Table 3, we made use of “Definition Validity”, which, according to [26], is met if the definition of a metric is sufficiently precise and unambiguous to allow the implementation of its collection in a special and deterministic way, implying full clarity and a lack of ambiguity. To evaluate whether the suggested metrics met the “Definition Validity” criterion, we consulted five specialists from the fields of software development and business management, who will also help us in the evaluation of ambiguity in FRs and qt-NFRs within the experimental section. The experts were three men and two women, software engineers with more than 10 years of experience in software development for Ecuadorian companies.

Based on system properties, an example of a qt-NFR would be “Registration speed”. In this case, the “Speed” property is being applied to the “registration” transaction. In the example in Figure 3, the “Robustness” property is being applied to a particular system.

An alternative to defining the title of a qt-NFR through the PROSE structure is to describe it directly using any system property metric. For example, “Transactions processed per second” could be the title of a qt-NFR, which, in this case, refers to the speed property of the system over the transaction operability action. However, this alternative has the disadvantage of not expressing (directly) what property of the system we are referring to.

Table 3. List of system properties and their suggested metrics.

System Property	Suggested Metrics	Reference
Speed	Transactions processed per second	[27]
	Response time to user and/or events	[28]
	Screen refresh time	[29]
Size	Lines of code	[30]
	Bytes (KB, MB, GB, PB, etc.)	[31]
	RAM memory capacity	[32]
Usability	Training time	[33]
	Number of help screens	[33]
	Task completion time	[33]
Reliability	Mean time between failures	[34]
	Probability of unavailability	[35]
	Failure occurrence rate	[36]
	Availability periods/hours	[37]
Robustness	Restart time between failures	[38]
	Percentage of events that generate failures	[39]
	Probability of data corruption after failures	[40]
Safety (Functional correctness)	Number of incorrect functionalities (“bugs”)	[41]
Security	Number of vulnerabilities	[42]
	Time to fix critical vulnerabilities	[43]
Resilience	Time to recover critical transactions	[44]
	Proportion of requests that were be lost during a disruptive incident	[45]
Compatibility	Number of supported operating systems	[46]
	Number of supported languages	[47]
Modularity	Number of times that the methods listed in a module are used by the other modules in the system	[48]
	Size (measured in non-commenting lines of code) of each module/package and the number of modules in the system	[49]
Complexity	The number of paths that are possible in a system	[50]
	Number of functions that are difficult to understand in a system’s code base	[51]

It is important to mention that Table 3 does not show all the properties that can be evaluated in a system but only presents the most popular ones. For example, in the field of the Internet of Things, you can have the following properties: scalability, real-time, timeliness, availability and reliability of service, security, and reusability [52]. For each of them, their respective metrics and values expected by the stakeholders must be defined.

Finally, requirements such as any programming language (java, Python), libraries and versions to use, and supported languages, among others required by the client, should be written in the constraint section of the requirement specification document. However, design or implementation details and additional restrictions on the software should be excluded from such a specification following IEEE STD 830-1998 [53].

3.3.2. Adding a Short Definition

In the “short definition” section of the SURE format, we express a synopsis of the system property and its value expected by the stakeholder. This can be seen in the example of Figure 4. This synopsis corresponds to a user-level requirement and should therefore be narrated in simple terms, without the use of technical computer words. More examples of short qt-NFR definitions are shown in Table 4.

Table 4. Examples of titles and short definitions of qt-NFRs.

Title	Short Definition
System robustness	The system must be operated within a maximum of 30 min after the failure occurs.
Transaction reliability	After a power outage, the system must allow purchase transactions to be carried out in less than 1 min. In addition, it must be available 24 h a day, 7 days a week.
System usability	A user of the system should not make more than three errors per day on average.
System compatibility	The system must be prepared for its respective use in Windows 7 or higher.

Something important to mention is that, in this short description, the expected values for the metrics that are part of that requirement must be indicated. In this way, we can avoid the definition of ambiguous and non-measurable qt-NFRs such as “The system must be easy to use”. That requirement refers to the usability property, but we cannot quantitatively validate compliance with that requirement. The correct thing would be to use a metric and indicate its expected value, such as “A user of the system should not make more than three errors per day on average”. Thus, by indicating an error limit value for a user, we can verify that the system is easy to use.

3.3.3. Adding Description Details

The “description and details of restrictions” part of the SURE format must provide technical information that allows to achieve those expected values defined in the previous section of the format (short definition). Examples of this section are presented in Table 5.

Table 5. Examples of titles and description details of qt-NFRs.

Title	Description Details
System robustness	The system must follow the master-slave replication model so that in the event of a failure, it can be operational in less than 30 min after the failure occurs.
Transaction reliability	The system must have both its .war artifact and its database replicated so that after a power outage, the system can continue operating in less than 1 min. In addition, the web application must be hosted on a cloud server so that it is available 24 h a day, 7 days a week.
System usability	The system interface must consider Norman’s principles for the location of the elements of each screen. In addition, 20 h of training must be provided to users in the use of the sales module operations. After this, a system user should not make more than three errors a day on average.
System compatibility	The system must be prepared for its respective use in any Windows 7 or higher environment, meeting the following minimum requirements: Screen resolution 800 × 600, or higher. RAM memory capacity: 4 GB or higher.

4. Results and Analysis

To assess whether the proposed CRUDE and PROSE structures help students define unambiguous FRs and qt-NFRs through the SURE format, 46 requirements specification documents from the RE courses offered at the University of Guayaquil between 2020 and 2022 were evaluated. Each document was developed as a course assignment by five or six students. Students had to document different cases of business systems such as the following:

- System to handle the sale of fruits, vegetables, soft drinks, alcoholic beverages, meats, personal hygiene, and home cleaning materials.
- System to manage advertising work orders.
- System to manage the distribution of drivers to vehicles to transport material from suppliers' warehouses to customers and manage payments.
- System to manage all the usual procedures of a travel agency: maintenance of accommodations, transportation companies, meals, complementary services, etc., always storing rates, characteristics, schedules, and other significant data.
- System to manage properties, which includes the management of both the fees paid by the owners, as well as all the expenses that occur and are of a common nature.
- System to manage ambulances, both requests for services and the assignment of ambulances, drivers, and doctors to each of those requests.
- System to manage the payments of municipal taxes related to the circulation of vehicles.
- System to manage activities within an automotive mechanics workshop.

Before and after the training class, which addressed the CRUDE and PROSE structures (proposed in this manuscript), they were required to create a minimum of thirty FRs and ten qt-NFRs in the manner shown in Figures 3 and 4.

Regarding the students, the selected sample corresponds to students enrolled in two groups of RE courses offered at the University of Guayaquil between 2020 and 2022. The RE course is taught in the second year of the Software Engineering degree. Normally, six to eight RE groups are opened each semester, each with a maximum of 35 students. Thus, the selected sample ranges from 25% to 33% of the RE courses offered each semester during the period 2020–2022. Furthermore, the majority of students are between the ages of 19–22. The distribution of men and women is usually equal.

Regarding the experts, five specialists from the fields of software development and business management assessed the ambiguity in FRs and qt-NFRs of the 46 requirements specification documents. To determine if any FR or qt-NFRs are ambiguous, the points described in Table 6 were checked by them. It is enough that a requirement does not comply with a criterion of its type (FR or qt-NFR) for it to be marked as ambiguous.

Table 6. Criteria used to determine if a requirement was ambiguous.

Type	Criteria
FR	Does not generate double interpretation?
	Does it define the application of any CRUD operation to any entity?
	Does it define the data types, lengths, and restrictions for each attribute involved in the requirement?
qt-NFRs	Is clearly described the system property being evaluated?
	Is the system property value expected by the stakeholder described?
	Are the technologies or techniques with which this requirement can be met described?

A requirement is classified as ambiguous if three or more experts have indicated it as ambiguous. In addition, they were asked to point out which part of the SURE format contained ambiguity. Below, we present the results obtained for both FRs and qt-NFRs.

4.1. Functional Requirements

4.1.1. Ambiguity in the Functional Requirements

The results of evaluating the FRs expressed on the requirement specification documents before explaining the CRUDE structure to students are displayed in Figure 5. It shows that on average, more than 200—or more than 80% of the requirements specified—have been deemed unclear by the majority of experts for each semester, whereas fewer than 60 requirements have been deemed clear by them.

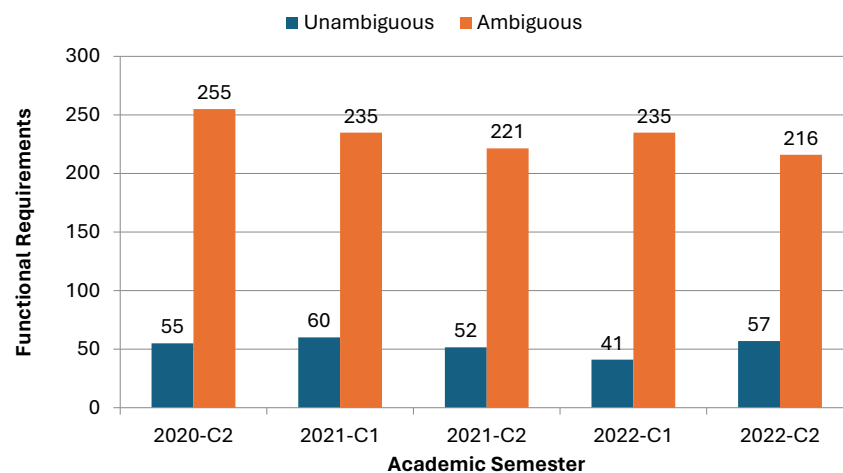


Figure 5. Evaluation of FRs collected before explaining the CRUDE structure to students during the period 2020–2022.

Conversely, Figure 6 displays the findings of the assessment of the FRs that were gathered following a class in which students corrected the requirements to comply with the CRUDE structure. It shows that, for each semester, the majority of the experts have determined that over 200 requirements are unambiguous, while less than 20 requirements—or less than 10% of the requirements—have been deemed unclear by them.

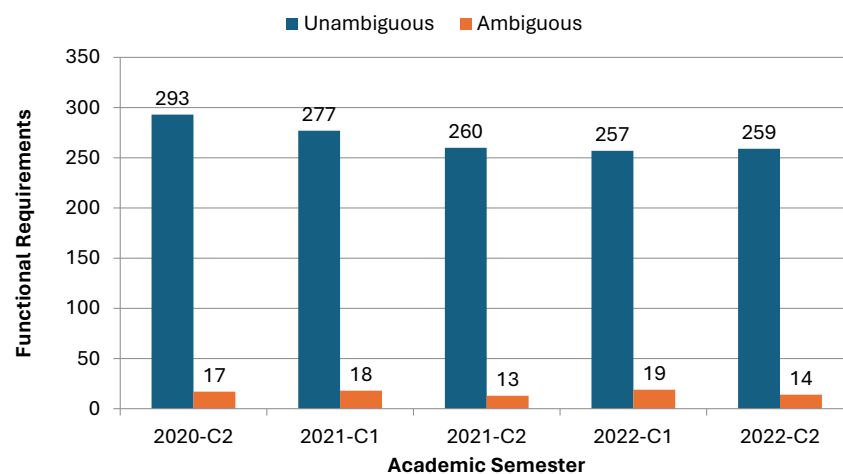


Figure 6. Evaluation of FRs collected after explaining the CRUDE structure to students during the period 2020–2022.

4.1.2. Format Sections with Ambiguity

Furthermore, on the FRs that were marked as ambiguous and collected before explaining the CRUDE structure, the experts were also requested to identify which part of the SURE format presented ambiguities. Figure 7 displays the findings. According to the findings it presents, the requirement title specification is where the majority of misunderstandings in FRs come from.

Additionally, concerning FRs that were marked as ambiguous after the session covering the CRUDE structure, experts were asked to identify which part of the SURE format presented ambiguities. Figure 8 displays the findings. According to the findings, the majority of ambiguity in FRs stems from the requirement title definition, just like in the prior instance.

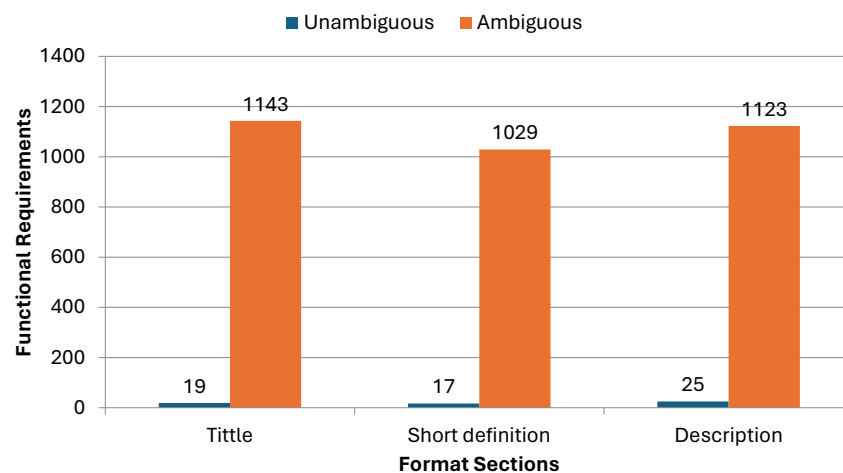


Figure 7. Evaluation of the SURE format sections on those FRs indicated as ambiguous before explaining the CRUDE structure.

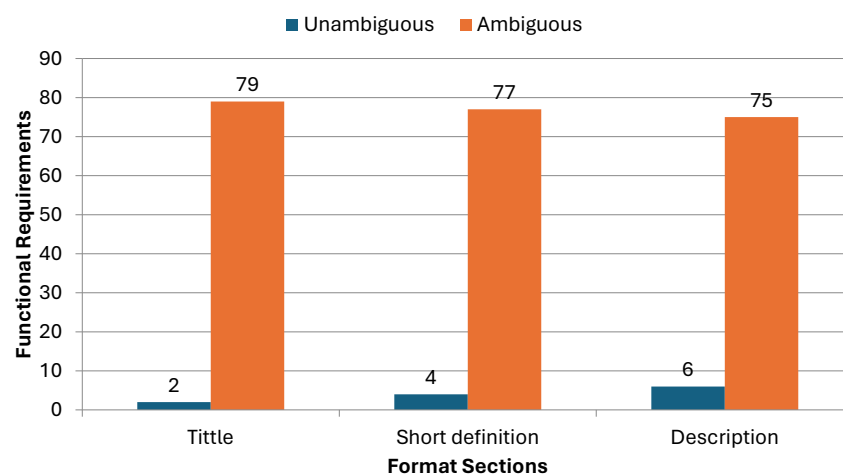


Figure 8. Evaluation of the SURE format sections on those FRs indicated as ambiguous after explaining the CRUDE structure.

The findings shown in Figures 7 and 8 enable us to conclude that, in the majority of cases, when a FR has been flagged as unclear, the requirement title is similarly unclear. In other words, vague FRs generally arise from a poor explanation of the requirement title. Conversely, it is considerably simpler to construct unambiguous FRs when the title is clear. To assist the student in developing clear FRs, the results emphasize the significance of creating effective requirement titles utilizing the CRUDE structure.

4.2. Quantitative Non-Functional Requirements

4.2.1. Ambiguity in the Quantitative Non-Functional Functional Requirements

The results of evaluating the qt-NFRs gathered before the training class that covered the PROSE structure are displayed in Figure 9. It shows that, on average, more than 80 requirements—or more than 90% of them—have been deemed unclear by the majority of experts for each semester, whereas fewer than 10 qt-NFRs have been deemed clear by experts.

Conversely, Figure 10 displays the findings of the assessment of the qt-NFRs of the projects that were gathered following a class in which students corrected the requirements to comply with the PROSE structure. It shows that each semester, the majority of the experts have determined that close to 90% of qt-NFRs are unambiguous, while less than 10 requirements—or close to 10% of them—have been deemed unclear by experts.

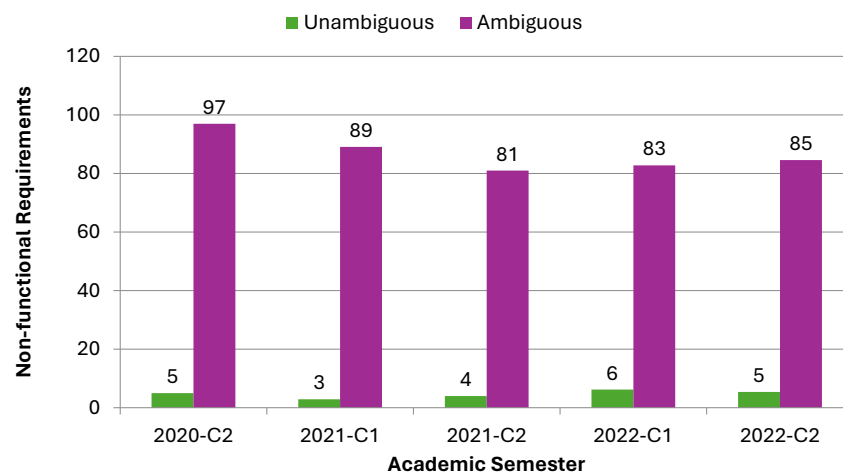


Figure 9. Evaluation of qt-NFRs collected before explaining the PROSE structure during the period 2020–2022.

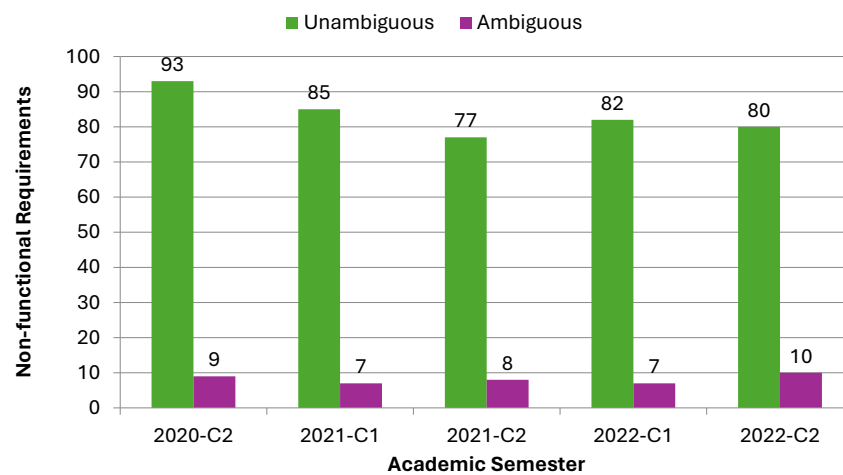


Figure 10. Evaluation of qt-NFRs collected after explaining the PROSE structure during the period 2020–2022.

4.2.2. Format Sections with Ambiguity

Additionally, experts were also asked to identify which part of the SURE format presented ambiguities for those qt-NFRs that were marked as ambiguous and collected before explaining the PROSE structure. Figure 11 shows the findings. According to the results, the short definition of qt-NFR is where most misunderstandings come from.

In addition, concerning the requirements that were flagged as ambiguous, the experts were asked to identify which part of the SURE format presented ambiguities for those qt-NFRs that were marked as ambiguous and collected after explaining the PROSE structure. Figure 12 displays the findings. According to the results, the majority of ambiguity in qt-NFRs stems from the requirement's short definition, just like in the prior instance.

The findings shown in Figures 11 and 12 enable us to conclude that, in the majority of cases, when a qt-NFR has been flagged as unclear, the short definition and description details of the requirement are also not clear. In other words, vague requirements generally arise from a poor definition of the metric values that must be achieved to comply with what the stakeholder requires. On the contrary, it is considerably easier to formulate a clear qt-NFR when the expected metric values of any system property are well-defined. To Help the student specify unambiguous qt-NFRs, the results emphasize the importance of defining the metric values expected by the stakeholder when considering the system properties and suggested metrics presented in Table 3 of the PROSE structure.

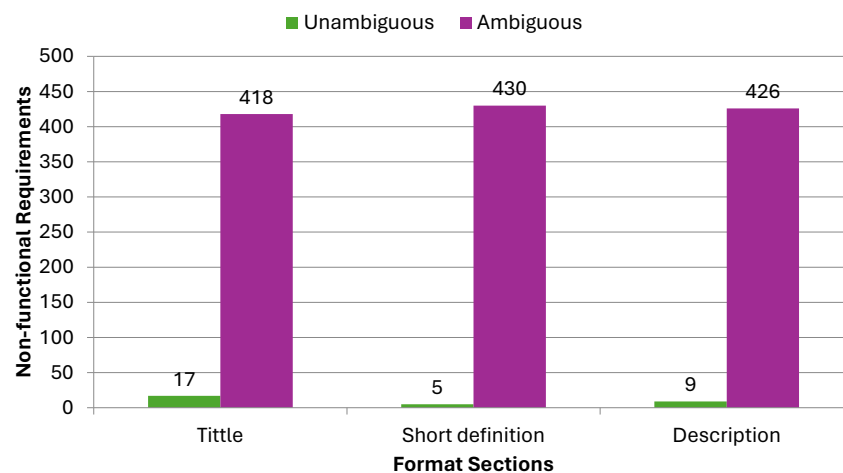


Figure 11. Evaluation of the SURE format sections on those qt-NFRs indicated as ambiguous before explaining the PROSE structure.

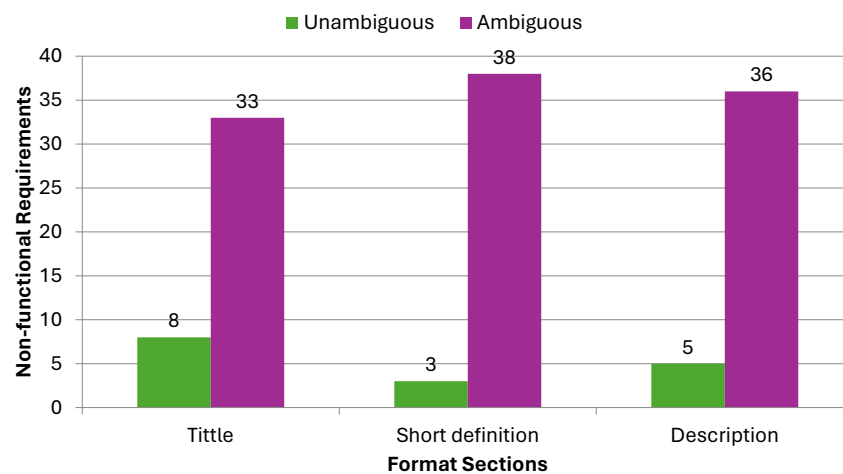


Figure 12. Evaluation of the SURE format sections on those requirements indicated as ambiguous after explaining the PROSE structure.

5. Conclusions

This manuscript proposes three structures to help students define unambiguous functional (FRs) and quantitative non-functional requirements (qt-NFRs). It entails completing the SURE structure, a format with three main sections: a title, a short definition, and a detailed description. To express unambiguous FRs, this study proposes a structure called CRUDE, which is based on CRUD operations performed on an entity. The format's description part also contains the fields or attributes that must be updated, registered, or used as a search field. Moreover, we propose the PROSE structure for specifying qt-NFRs. It is based on the application of properties to computer systems. In it, it is crucial to indicate the metric values that must be achieved to satisfy the expectations of the stakeholder.

In total, 46 requirement specification documents from the University of Guayaquil's RE courses taught between 2020 and 2022 were taken into consideration to assess the improvement in the definition of FRs and qt-NFRs while using the proposed structures. To evaluate its applicability in business systems, five experts from the fields of software development and business management assessed the ambiguity in FRs and qt-NFRs of the requirements specification documents. The results demonstrated that the percentage of unambiguous FRs and qt-NFRs decreased from more than 80% to around 10% when the structures recommended in this study were put into practice. Moreover, much of the uncertainty in FRs originates from the requirement title specification. Nonetheless, it is

simpler to define clear FRs when the title is clear. On the other hand, much of the ambiguity in qt-NFRs originates in the specification of their short definition. However, it is easier to define qt-NFRs when the values of the suggested metrics are defined within the PROSE structure. Overall, the results show how important the three structures presented in this manuscript are when helping the student define unambiguous requirements.

The creation of serious games that enable interactive learning of the structures suggested in this paper may be taken up as future research. Furthermore, we leave as subsequent work the study of structures that facilitate the unambiguous definition of qualitative non-functional requirements (ql-NFRs).

Author Contributions: All the authors have contributed to the work presented in this paper. Conceptualization, F.P.-B.; methodology, F.P.-B.; validation, F.P.-B., R.C.-Q., J.B.-M., L.V.-C., M.R.-W. and M.I.G.-S.; investigation, F.P.-B.; writing—original draft preparation, F.P.-B.; writing—review and editing, F.P.-B., R.C.-Q., J.B.-M., L.V.-C., M.I.G.-S. and M.R.-W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to express their gratitude to the experts of the requirements specification documents elaborated by students of the Software Engineering degree at the University of Guayaquil for their voluntary participation in the experiment.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Laplante, P.A.; Kassab, M. *Requirements Engineering for Software and Systems*; Auerbach Publications: Boca Raton, FL, USA, 2022.
2. Ryan, M.J.; Wheatcraft, L.S.; Dick, J.; Zinni, R. On the definition of terms in a requirements expression. In *INCOSE International Symposium*; Wiley Online Library: Hoboken, NJ, USA, 2015; Volume 25, pp. 169–181.
3. Durán, A.; Bernárdez, B.; Toro, M.; Corchuelo, R.; Ruiz, A.; Pérez, J. Expressing customer requirements using natural language requirements templates and patterns. In *Proceedings of the IMACS/IEEE CSCC'99 Proceedings*, Athens, Greece, 4–8 July 1999.
4. Wiegers, K.E.; Beatty, J. *Software Requirements*; Pearson Education: London, UK, 2013.
5. Asadabadi, M.R.; Saberi, M.; Zwikael, O.; Chang, E. Ambiguous requirements: A semi-automated approach to identify and clarify ambiguity in large-scale projects. *Comput. Ind. Eng.* **2020**, *149*, 106828. [\[CrossRef\]](#)
6. Umber, A.; Bajwa, I.S. Minimizing ambiguity in natural language software requirements specification. In *Proceedings of the 2011 Sixth International Conference on Digital Information Management*, Melbourne, Australia, 26–28 September 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 102–107.
7. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.V.; Batista-Navarro, R.T. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 55. [\[CrossRef\]](#)
8. Mokos, K.; Nestoridis, T.; Katsaros, P.; Bassiliades, N. Semantic Modeling and Analysis of Natural Language System Requirements. *IEEE Access* **2022**, *10*, 84094–84119. [\[CrossRef\]](#)
9. González, S.E.; Reyes, J.R.; Cedeño, D.C.; Azcuy, R.A.; Diez, H.G. Herramienta de procesamiento del lenguaje natural para la detección de ambigüedades en requisitos de software en español. *Rev. Cuba. Cienc. Inform.* **2023**, *17*, 185–204.
10. López-cortés, N.; Horno Chéliz, C. La Naturaleza de la Ambigüedad Léxica. Un Estudio sobre los Sustantivos en Español. Ph.D. Thesis, Universidad de Zaragoza, Zaragoza, Spain, 2021. Available online: <https://zaguan.unizar.es/record/108451> (accessed on 10 May 2024).
11. Werner, C.; Li, Z.S.; Ernst, N.; Damian, D. The lack of shared understanding of non-functional requirements in continuous software engineering: Accidental or essential? In *Proceedings of the 2020 IEEE 28th International Requirements Engineering Conference (RE)*, Zurich, Switzerland, 31 August–4 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 90–101.
12. Ashfaq, F.; Bajwa, I.S.; Kazmi, R.; Khan, A.; Ilyas, M. An intelligent analytics approach to minimize complexity in ambiguous software requirements. *Sci. Program.* **2021**, *2021*, 6616564. [\[CrossRef\]](#)
13. Ezzini, S.; Abualhaija, S.; Arora, C.; Sabetzadeh, M.; Briand, L.C. Using domain-specific corpora for improved handling of ambiguity in requirements. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, Madrid, Spain, 22–30 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1485–1497.
14. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F. Automated checking of conformance to requirements templates using natural language processing. *IEEE Trans. Softw. Eng.* **2015**, *41*, 944–968. [\[CrossRef\]](#)
15. Atoum, I.; Baklizi, M.K.; Alsmadi, I.; Ootom, A.A.; Alhersh, T.; Ababneh, J.; Almalki, J.; Alshahrani, S.M. Challenges of software requirements quality assurance and validation: A systematic literature review. *IEEE Access* **2021**, *9*, 137613–137634. [\[CrossRef\]](#)

16. Fantoni, G.; Coli, E.; Chiarello, F.; Apreda, R.; Dell’Orletta, F.; Pratelli, G. Text mining tool for translating terms of contract into technical specifications: Development and application in the railway sector. *Comput. Ind.* **2021**, *124*, 103357. [\[CrossRef\]](#)
17. Glinz, M. On non-functional requirements. In Proceedings of the 15th IEEE International Requirements Engineering Conference (RE 2007), Delhi, India, 15–19 October 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 21–26.
18. Glinz, M. Rethinking the notion of non-functional requirements. In Proceedings of the Third World Congress for Software Quality, Munich, Germany, 15–21 May 2005; Volume 2, pp. 55–64.
19. Zhou, Z.; Zhi, Q.; Morisaki, S.; Yamamoto, S. An evaluation of quantitative non-functional requirements assurance using ArchiMate. *IEEE Access* **2020**, *8*, 72395–72410. [\[CrossRef\]](#)
20. Ribeiro, Q.A.; Ribeiro, M.; Castro, J. Requirements engineering for autonomous vehicles: A systematic literature review. In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, Virtual, 25–29 April 2022; pp. 1299–1308.
21. Wilson, W.M. Writing effective natural language requirements specifications. *Crosstalk J. Def. Softw. Eng.* **1999**, 16–19. Available online: <https://api.semanticscholar.org/CorpusID:59716545> (accessed on 21 May 2024).
22. Pohl, K. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Examination Level-IREB Compliant*; Rocky Nook, Inc.: San Rafael, CA, USA, 2016.
23. Mavin, A.; Wilkinson, P.; Harwood, A.; Novak, M. Easy approach to requirements syntax (EARS). In Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 30 August–4 September 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 317–322.
24. Kopczyńska, S.; Nawrocki, J. Using non-functional requirements templates for elicitation: A case study. In Proceedings of the 2014 IEEE 4th International Workshop on Requirements Patterns (RePa), Karlskrona, Sweden, 26 August 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 47–54.
25. Kopczyńska, S.; Nawrocki, J.; Ochodek, M. An empirical study on catalog of non-functional requirement templates: Usefulness and maintenance issues. *Inf. Softw. Technol.* **2018**, *103*, 75–91. [\[CrossRef\]](#)
26. Meneely, A.; Smith, B.; Williams, L. Validating software metrics: A spectrum of philosophies. *ACM Trans. Softw. Eng. Methodol.* (TOSEM) **2013**, *21*, 24. [\[CrossRef\]](#)
27. Bernstein, P.A.; Newcomer, E. *Principles of Transaction Processing*; Morgan Kaufmann: Burlington, MA, USA, 2009.
28. Joseph, M.; Pandya, P. Finding response times in a real-time system. *Comput. J.* **1986**, *29*, 390–395. [\[CrossRef\]](#)
29. Kennedy, A.; Baccino, T. The effects of screen refresh rate on editing operations using a computer mouse pointing device. *Q. J. Exp. Psychol.* **1995**, *48*, 55–71. [\[CrossRef\]](#)
30. Li, H.F.; Cheung, W.K. An empirical study of software metrics. *IEEE Trans. Softw. Eng.* **1987**, *6*, 697–708. [\[CrossRef\]](#)
31. Jones, A.K.; Schwans, K. Task forces: Distributed software for solving problems of substantial size. In Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 17–19 September 1979; pp. 315–330.
32. Smith, A.J. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. *IEEE Trans. Softw. Eng.* **1978**, *SE-4*, 121–130. [\[CrossRef\]](#)
33. Albert, B.; Tullis, T. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*; Newnes: Oxford, UK, 2013.
34. Morgan, D.E.; Taylor, D.J. Special feature a survey of methods of achieving reliable software. *Computer* **1977**, *10*, 44–53. [\[CrossRef\]](#)
35. Trivedi, K.S.; Kim, D.S.; Ghosh, R. Resilience in computer systems and networks. In Proceedings of the 2009 International Conference on Computer-Aided Design, San Jose, CA, USA, 2–5 November 2009; pp. 74–77.
36. Goseva-Popstojanova, K.; Trivedi, K.S. Failure correlation in software reliability models. *IEEE Trans. Reliab.* **2000**, *49*, 37–48. [\[CrossRef\]](#)
37. Ford, D.; Labelle, F.; Popovici, F.I.; Stokely, M.; Truong, V.A.; Barroso, L.; Grimes, C.; Quinlan, S. Availability in globally distributed storage systems. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), Vancouver, BC, Canada, 4–6 October 2010.
38. Sayani, H.H. *A Decision-Model for Restart and Recovery from Errors in Information Processing Systems*; University of Michigan: Ann Arbor, MI, USA, 1973.
39. Maxion, R.A.; Olszewski, R.T. Improving software robustness with dependability cases. In Proceedings of the Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 98CB36224), Munich, Germany, 23–25 June 1998; IEEE: Piscataway, NJ, USA, 1998; pp. 346–355.
40. Nagesh, P.; Srinivasu, N. Modelling a secure framework for data verification and integrity in cloud environment. *Int. J. Syst. Assur. Eng. Manag.* **2022**, 1–12. [\[CrossRef\]](#)
41. Rosenberg, L.; Hammer, T.; Shaw, J. Software metrics and reliability. In Proceedings of the 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 4–7 November 1998.
42. Medeiros, N.; Ivaki, N.; Costa, P.; Vieira, M. Software metrics as indicators of security vulnerabilities. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 216–227.
43. Bekmagambetova, G.; Polukhin, A.; Evdokimov, V.; Kasmin, D.; Dmytriienko, O. Algorithmic means of ensuring network security and websites: Trends, models, future cases. *Amazon. Investig.* **2023**, *12*, 149–163. [\[CrossRef\]](#)
44. Fang, Y.P.; Pedroni, N.; Zio, E. Resilience-based component importance measures for critical infrastructure network systems. *IEEE Trans. Reliab.* **2016**, *65*, 502–512. [\[CrossRef\]](#)

45. Curtis, B. How Do You Measure Software Resilience? 2019. Available online: <https://bit.ly/3Poz79o> (accessed on 21 May 2024).
46. Poljak, R.; Pošćić, P.; Jakšić, D. Comparative analysis of the selected relational database management systems. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1496–1500.
47. Cleary, A.; Kohn, S.; Smith, S.G.; Smolinski, B. *Language Interoperability Mechanisms for High-Performance Scientific Applications*; Technical Report; Lawrence Livermore National Lab. (LLNL): Livermore, CA, USA, 1998.
48. Sarkar, S.; Kak, A.C.; Rama, G.M. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Trans. Softw. Eng.* **2008**, *34*, 700–720. [[CrossRef](#)]
49. Emanuel, A.W.R.; Wardoyo, R.; Istiyanto, J.E.; Mustofa, K. Modularity Index Metrics for Java-Based Open Source Software Projects. *Int. J. Adv. Comput. Sci. Appl.* **2011**, *2*. [[CrossRef](#)]
50. McCabe, T.J. A complexity measure. *IEEE Trans. Softw. Eng.* **1976**, *SE-2*, 308–320. [[CrossRef](#)]
51. Antinyan, V.; Staron, M.; Derehag, J.; Runsten, M.; Wikström, E.; Meding, W.; Henriksson, A.; Hansson, J. Identifying complex functions: By investigating various aspects of code complexity. In Proceedings of the 2015 Science and Information Conference (SAI), London, UK, 28–30 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 879–888.
52. Trilles, S.; González-Pérez, A.; Huerta, J. An IoT platform based on microservices and serverless paradigms for smart farming purposes. *Sensors* **2020**, *20*, 2418. [[CrossRef](#)]
53. Institute of Electrical and Electronics Engineers. *IEEE Guide to Software Requirements Specifications*; IEEE: Piscataway, NJ, USA, 1998.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.