A Method to Evaluating Consistency, Completeness and Correctness in Evolution Requirements

Edneuci Denise Audacio[1], Katia Romero Felizardo[1], Luiz Gustavo Ferreira Aguiar[3], Rebeca Teodoro da Silva[3], and Elias Canhadas Genvigir[2]

[1] Federal University of Technology – Parana - Campus Cornélio Procópio - Brazil
[2] Federal University of Technology – Parana - Campus Londrina - Brazil
{katiascannavino,elias}@utfpr.edu.br
[3] Tribunal de Justiça do Estado do Paraná - TJPR
{luiz.aguiar, rebeca.teodoro}@tjpr.jus.br

Abstract. Changes in the domain in which a specific software was developed can generate a demand for new requirements known as requirements evolution. However, it is expected that these will be specified in a consistent, complete and correct manner. This work defines a method that makes it possible to assess the consistency, completeness and correctness of requirements defined during the software evolution. The developed method is composed of two phases: (1) analysis of units of information, i.e., the analysis of each requirement and its domain; and (2) analysis of these items through indicators for consistency and completeness. For verification purposes, the method was applied through a case study in a software company and, the results presents positive indicators for the improvement of quality in requirements evolution. The project, object of this study, originally had a high rework load, that is, correction in the codification of the requirements of the case study. Through the application of the method, it was possible to identify that most of the evaluated requirements, which presented inconsistency or completeness problem, were associated with rework efforts.

Keywords: Requirements Engineering · Requirements Evolution · Requirements Domain · Requirements Assessment.

1 Introduction

It is known that the basis for a good software development is related to the quality of the requirements attributes [4] [16]. Identifying the right requirements is an arduous and iterative task, where requirements engineers must respond to the dual challenge of discovering and formalizing the wants and needs that customers are usually able to define only in a confused and incomplete way [10].

Initially, the requirements are raised through dialogues with customers, being, in most cases, represented in natural language, in which the customer exposes his/her anxieties, needs and his/her perception on the problem, while the

requirements engineer employs techniques with the objective to identify the requirements and aspects related to the domain [8].

In the same sense, Kamalrudin [12] and Ferrari [8] suggests that the requirements written in natural language are by definition ambiguous, and may induce inaccuracy, inconsistency and incompleteness. However, it is expected that the requirements of a project are clearly defined in a consistent, complete and correct way, in order to enable the comprehension and understanding of the requested needs, as well as the validation of these by the client with the certainty that they were correctly attended to.

The quality elements of requirements described in natural language can be evaluated through consistency, completeness and correctness considering the information domain of these requirements [22].

The objective of this study is to define a method that allows to assess the consistency, completeness and correctness of requirements defined in Service Orders (S.O.), which were previously raised in a software development company. The presented method is composed of two phases based on the analysis of the Units of Information (U.I.), that constitute each requirement, and its information domain. After the definition, the method was applied to real requirements and its results show positive indications for evaluating quality parameters in requirements.

Other studies have been carried out to assess the consistency, completeness and correctness of the software requirements. Saito et al. [17] propose a basic methodology based on a checklist and reports to verify and validate the consistency and correctness of the requirements. In turn, the Case-BasedReasoning (CBR) technique developed by Jani et al. [11] aims to validate whether the standards and requirements procedures are being followed within the requirements specification phase, but the technique only observes whether the verified requirements reached or not the quality attributes, do not cause failures, defects or even thelack of requirements. On the other hand, Saito et al. [17] propose that after verification, a note is made on items that need improvement.

Muriana et al. [13] present the QualiCes method (Quality via Consistency in Software Specifications) which aims to verify the consistency between the documents. However, the method is limited to analyzing the use cases that have prototypes, it is a restriction, since it is not always possible to produce prototypes from the use cases.

Genova et al. [10] emphasize the consistency and completeness of the requirement quality attributes by analyzing the textual quality of requirements, using formal requirements documents as input data. In this same perspective, Soares and Moura [19] present the use of a writing methodology for the requirements specification, using the ERS-EDITOR software, which is based on the construction of an expanded lexicon of language.

To reduce the inconsistency generated by the use of natural language, Fockel and Holtmann [9] use a requirements documentation approach that applies to the so-called requirements patternsas Controlled Natural Language (CNL). CNL re-

stricts the expressiveness of natural language (NL) to allow automatic processing of requirements, while still keeping them comprehensible to all interested parties.

In order to restrict the inconsistency and incorrectness of the requirements extracted from the use of natural language, Palomares et al. [14] present the use of a tool called PABRE, which is based on requirements elicitation standards. The tool PABRE consists of accumulating experience of requirements with similar purposes and, from them, deducing a refined model of well-formed requirements and with spaces reserved for extension points.In addition, each model is enriched with detailed metadata, which is structured according to an organization based on form templates.

It is worth mentioning that the method proposed in this article proposes an evaluation of the consistency, completeness and correctness of the requirements registered in the requirements specification artifacts, having as a principle, the fragmentation of requirements into units and their analysis together with their information domain.

2 Concepts

The consistency of a requirement refers to situations in which a specification does not contain internal contradictions, being a property of a certain knowledge that implies in mutually exclusive declarations and without confrontations of the terminology [18] [5] [21].

In many cases, inconsistency between requirements can be related to incorrect actions, conflict between requirements, weak dependencies and even the lack of ability of users to describe the real need, as well as the lack of understanding of analysts [12] [15]. In other words, depending on the specified rule or relationship being considered, any defect in a specification, such as a lack of clarity and understanding of the writing, may be termed inconsistency. Therefore, these complications often represent incomplete requirements [12].

Among the reasons for the occurrence of inconsistencies, one can also includes different: linguistic uses, development strategies and points of view of the participants, and the degree of overlap that exists in the areas of concern of different stakeholders, which may also lead to incomplete requirements [21].

The consistency of a requirement is directly related to its completeness, since for a requirement to be complete, it must contain at least three characteristics, namely: 1) No information should be left as undeclared, undetermined or unrelated to the domain; 2) The information must not contain objects, entities or indefinite terms, that is, each operation or condition must be constructed using syntax and semantic rules and 3) No information must be absent, that is, all parts or units must be present and each part or unit must be fully developed [3] [6] [18].

If a requirement is incomplete by any other definition, developers are likely to make assumptions about the intended behavior, and those assumptions can lead to misunderstandings and, therefore, a product that does not meet the requested need [6]

There is a need for the correct description, without bias, of requirements because attempts to increase the completeness of a requirements specification can impact the decrease in consistency and, therefore, affect the correction of the final product. On the other hand, increasing the consistency of the requirements tends to reduce the completeness and consequently decrease the correction [20]. Regarding correctness it is considered that a requirement is correct if it is declared as consistent and complete in relation to the real needs of users [18].

Incorrect requirements can occur when the requirements do not accurately reflect the facts or erroneous predictions about future states. Zowghi and Gervasi [22] refer to the correction of requirements as being a combination of consistency and completeness, being also often defined by the customer as the satisfaction of certain commercial objectives.

It is observed that considerations to be made about consistency and completeness go through the analysis of the domain in which the requirements are found. In this way, a considerable part of the requirements engineering efforts are in analyzing and representing the existing information in the knowledge domain. In other words, a good part of the requirements of a software constitute a subset of information from it is domain and the analysis requirements quality elements is intrinsically linked to the understanding of the domain.

One of the techniques used to better understand the domain is called domain modeling, which can provide an increase in communication between customers and requirements engineers and, consequently, facilitate the understanding of the real need of the stakeholder [21]. In many cases, modeling the application domain is adopted to develop a conceptual model in order to direct analysts directly to the customer's real needs.

Another important role of domain modeling is associated with changes in requirements. Some authors suggest that in iterative development cycles, with each increment in requirements, these cannot contradict the domain so that consistency and completeness are maintained [2] [15] [18] [22] [21]. In this context, requirements are rarely defined at once, and in many cases, the requirements are achieved through the progressive evolution of a previous version of the same requirements or even the evolution of the software, in order to reflect a greater understanding of customer needs [1] [5] [7] [21]. In the same way, the domain evolves, since there will be a deepening of the client's real needs.

Another aspect is that after the completion of the software development, changes in the domain may occur, generating the need for evolution in the software. Such needs are presented as requirements evolution. The role of the software's evolution requirements is to determine which relationships they wish to maintain between the elements of a certain domain [12]. In an already completed software, the collection of the requirements evolution does not always occur as in the elicitation of a new product. Customer service channels such as chats, call center services, e-mails and other distance services are often used. One way of recording these evolution requirements, received by the service channels, is through Service Orders (O.S.) in which the requests from stakeholders are the inputs to the beginning of the process of specifying these requirements.

An S.O. may have the following information: 1) Customer request; 2) Technical identification and analysis for developers; 3) Identification of the module that needs correction or implementation; 4) Detailing of the activities involved; 5) Control of identification of those involved in the analysis process, development and others involved in the request; 6) Indication of estimated and realized hours for the development of the request; 7) Project and version information that will be contemplated the request of the correction or implementation; 8) Images or prototyping to assist in the process of understanding the request for correction or implementation and finally; 9) The instruction for the user regarding the implementation or correction carried out.

3 Method

As can be observed in Figure 1, to assess completeness, consistency and correctness in the requirements evolution, presented in S.Os, a method was developed based on two phases: I) Analysis of the sentences described in the Service Order and II) Assessment of consistency, completeness and correctness in the sentences.

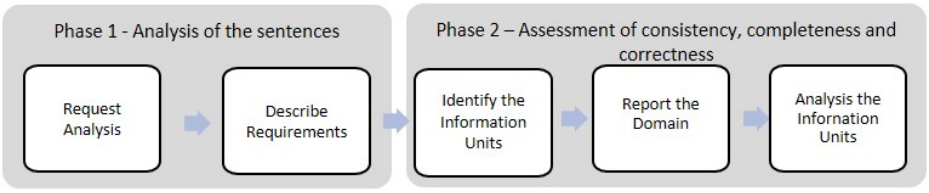


Fig. 1: Conceptual phases of the method and it's main activities

3.1 First Phase of the Method - Analysis of the Sentences

In an S.O. the request is written in natural language, with the transcription occurring fully as described by the stakeholder. As shown in Figure 2, in the first phase, the description of the request is analyzed, in order to understand and interpret the stakeholder's need. In the sequence, the request is broken down into requirements. After this step, a description of the request domain is elaborated to allow the analysis of the Units of Information that are extracted from the requirements. A requirement may consist of one or more units of information and an S.O. may have one or more customer requests. Each request can generate one or more units of information that can compose one or more requirements as well as additional information to the domain.

The term Units of Information (U.I.) refers to two types of information that the customer may wish to articulate throughout the dialogues: 1) the software requirements and 2) aspects related to the softwaredomain. Therefore, the articulation of an units of information is the fragment of speech in which the customer expresses a software requirement or additional information to the domain [1][8].

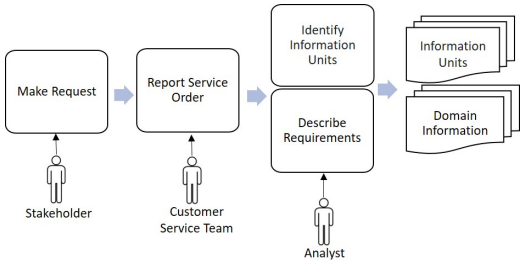


Fig. 2: Relationship between S.O., Units of Information and requirements

The identification of an Units of Information is carried out in three phases of a mental structure: interpretation, acceptance and accessibility[1]. When completing the model reading, it should be noted whether the stakeholder transmits an unit of information, which represents any information related to the software requirements or domain knowledge. This Units of Information must be interpretable, acceptable and accessible by the requirements analyst, so that the communication takes place efficiently [1].

In the interpretation phase, the possible meanings for each Units of Information received are assigned by the requirements analyst or engineer. In the acceptance phase, the analyst selects the interpretations received in his/her mental structure, whether or not they can be accepted. Finally, the accessibility process is analyzed and verified, where the results produced by the requirements analyst are consistent with the information requested by the user [1] [8]. These phases can be seen in Figure 3, in which the Process Model for accessing an Units of Information is presented.

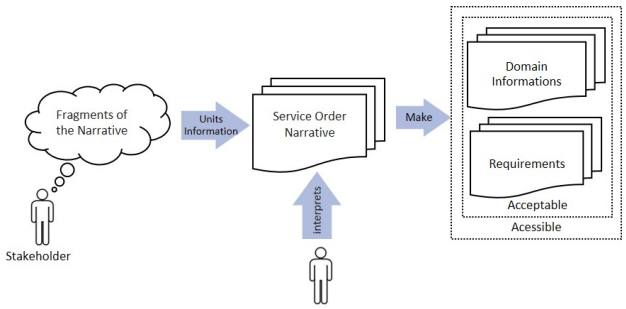


Fig. 3: Analisys of Units of Information

3.2 Second Phase of the Method - Assessment of Consistency, Completeness and Correctness

The second phase of the method consists of the assessment of the consistency, completeness and correctness of the requirements occurring by the evaluation of these items in the units of information that compose the requirements. To this end, a set of questions was elaborated, three of consistency and three of completeness. The questionnaire was based on the literature [1] [3] [5] [6] [8] [18] [21] [22].

The questions related consistency analysis are: 1) The units of information must not be contradictory with each other or with the domain; 2) Declarations must be mutually exclusive; 3) There should be no confrontation of terminology or terminology that causes doubts or misinterpretation [1] [5] [6] [18] [21].

The questions for completeness analysis are: 1) All information regarding the requirement must be declared in the domain; 2) The information must not contain objects, entities or indefinite terms; 3) No information should be missing [3] [6]. If a unit of information and/or need is consistent and complete, it will be correctly declared [6] [18] [21] [22].

In turn, the assessment of the requirement correctness occurs through consistency and completeness indicators. A requirement is correct if it is stated as consistent and complete, that is, the combination of both. In this way, initially, consistency and completeness are evaluated, and the requirement will be correct if these two indicators are positive, otherwise the requirement is considered incorrect.

Each unit of information is evaluated according to the criteria established by the questions. As an example for the application of the method, a S.O. was considered with a request made by a customer, separated in requirements and by its units of information and the description of the request domain (see Table 1).

Table 1: Example of the first phase of the method

Customer Request		
When registering tire brands, the software is allowing the inclusion of special characters and should allow changes to be made to the registration of a tire brand if there are no tires with a history of movement. The software displays an "ERROR" message when deleting a tire.		
Requirements	**Detailing**	**Units of Information**
RE1	On the Tire Brand Registration screen, the brand field accepts special characters	U.I.1
RE2	The system should not allow a tire record to be changed if it has not already been associated with any other record in the system.	U.I.2
RE3	When trying to delete a brand, which has already been used, the system should display a message whose title is "ERROR".	U.I.3
Domain details – Tire Registration		
Tire information is used by the purchasing software and carriers' maintenance. Therefore, it is necessary to perform the correct registration of the tires and observe various information. Every tire has a unique identification number and cannot be duplicated. The tires have a brand (manufacturer), model and dimension, and each brand can have several models. The description of the brand name can be simple or composed, and must not contain special characters such as: @, #, $, %, &, *, (, or space. However, the description of the tire model name can be simple, composed and can have letters, numbers and symbols: + and /. A tire has a dimension that is the size of the tire characterized by width, height, type of construction and rim. composed of letters, numbers and the symbols of / and points. When registering a tire, it is linked with its respective brand, model and dimension. If a tire is linked to a vehicle, its registration must not allow changes and exclusion and if the user tries to make changes or exclude the tire, the system should provide an "ALERT" message that the tire is already in a vehicle. The software should allow alteration and exclusion of the brand, model and dimension of the tire as long as there is no tire movement in vehicles and must not accept alteration of the tire number, even if there is no movement, in case of error you must delete the registration and redo it.		

It is observed that, for this example, each Requirement has one Unit Information. Thus, RE1 is an abbreviation to Requirement 1 which, in turn, has a unique Unit Information - U.I.1. However, a requirement can be composed of more than one unit information.

After the domain description, the Units of Information are analyzed for consistency and completeness issues. It was observed that, for the example presented in Table 2, each requirement contains a single U.I.

Table 2: Analysis of the Units of Information contained in the requirements of Table 1

Units	**Consistency Questions**			**Consistency Result**	**Completeness Questions**			**Completeness Result**	**Correctness Result**
	Q1	**Q2**	**Q3**	**U.I.**	**Q1**	**Q2**	**Q3**	**U.I.**	
U.I.1	✗	✓	✓	Inconsistent	✓	✓	✗	Incomplete	Incorrect
U.I.2	✓	✓	✓	Consistent	✓	✓	✓	Complete	Correct
U.I.3	✗	✓	✗	Inconsistente	✓	✗	✗	Incomplete	Incorrect

In this case, each U.I. make up an only requirement. Regarding consistency, the first question presents "the requirements cannot be contradicting each other nor with the domain". However, the units of informationU.I.1 and U.I.3 showed inconsistency with the domain. Concerning U.I.1 in the description of the domain, it was stated that in the registrations of brand and model of tires they cannot contain special characters, and for U.I.3 there is a mention in the domain for an alert message in case of attempts of alteration or exclusion, which makes the unity to be contradictory with the domain. Regarding the second question in which "the declarations must be exclusive", the three units of information were declared to be consistent because there were no undefined declarations, even though U.I.1 and U.I.3 were contradictory to the domain. And finally, regarding question number three, which stated that "there should be no confrontation of terminology or terminology that causes doubts or misinterpretation" U.I.3 caused doubt or misinterpretation and the others were considered consistent.

In the application of the questions regarding completeness, the first question recommended that "all information referring to the requirement must be declared in the domain", i.e., all units of information were classified as complete because the domain presents all the information described in the requirements. In question number two it was affirmed that "the requirement cannot contain indefinite terms or objects". In what concerns it, U.I.3 presented the term "ERROR" which was not defined, while the other units were classified as complete because they did not have indefinite terms. Moreover, question three showed that "no information should be missing". It was observed that U.I.1 had missing information, since the declaration of what a special character is, is incomplete, both in the requirement and in the domain, and U.I.3 did not describe the term "ERROR" and it is also not presented in the domain.

Only requirement two was considered correct while requirements one and three are considered incorrect because their units of information presented, in some of the evaluated issues, inconsistency or lack of completeness.

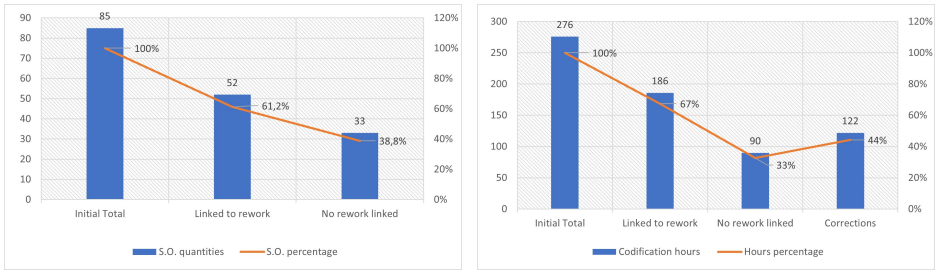
4 Applying the Method

In order to evaluate the method application, the analysis was carried out in real cases in a company. The company, object of this study, develops software for the transport and logistics area and has 13 employees, including programmers, analysts and managers, four developers, a project manager, two quality analysts, a deployment analyst, a support manager and four technicians in the support department. It was observed that the company had a constant need for corrections and coding rework in requests for new implementations submitted by customers. Thefore, the objective was to understand the quality of the generated S.O. to identify whether the coding rework could originate from the quality of the S.O. descriptions. As an implementation process, new requests are collected by chat or telephone and described in S.Os. A new integration or software module is initiated after collecting enough requests to generate a development interaction lasting a maximum of 60 days.

This study selected, in a period of 60 days, 85 S.Os., related to a project of modifications, corrections and new demands, in a system in the area of logistics, which generated a total of 276 hours of coding. As summarized in Table 3 and Figure 4, after the implementation and acceptance tests, with the users, it was identified that 52 S.Os. (61.2%) were associated with some type of correction and only 33 S.Os. (38.8%) had no association with correction. The corrections generated, due to failure to meet customer expectations or implementation errors, generated an additional of 122 hours of coding work totaling 398 hours on the project, i.e., an increase of 44% hours of rework over time initial coding of the Project.

Table 3: Quantitative of the S.Os. considered for testing the method

Service Order	**S.O. Quantity**	**S.O. (%)**	**Coding hours**	**Hours(%)**
Initial total	85	100%	276	100%
Linked to rework	52	61.2%	186	67%
Not reworked	33	38,8%	90	33%
Corrections	0	10%	122	44%



(a) quantitative of S.O linked or not to rework (b) Result of Consistency issues

Fig. 4: Individual result of the evaluation questions of the Units of Information in the second phase of the method.

The 85 S.Os. generated 99 requirements that were analyzed as U.I. during the first phase of the method. The goal was to indentify and assess whether the units of information are interpretable, acceptable and accessible by the requirements analyst, obtaining 151 U.I. For each S.O., the domain to be used to analyze the questions of the second phase of the method was described.

The second phase consisted of applying the questionnaire, with questions of completeness and consistency, for each of the U.I. As an overall result of this analysis, it was observed that 89 units were consistent, 118 were incomplete which, consequently, led to non-correctness. Thus, 118 units were incorrect and only 33 were correct, as can be seen in Figure 5.

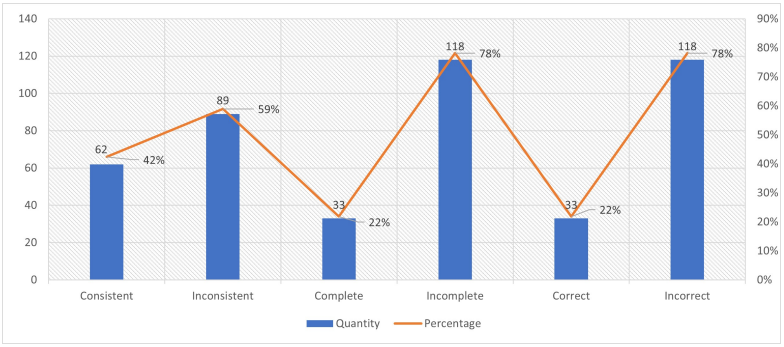
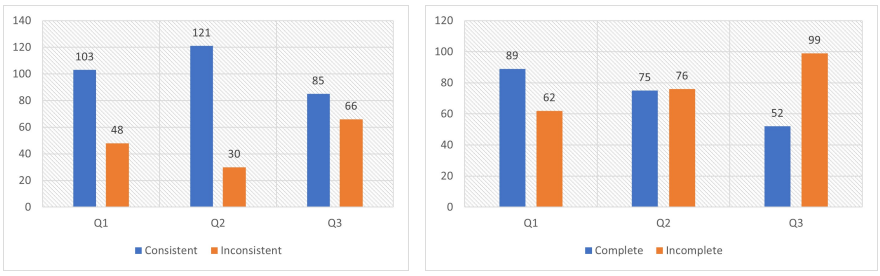


Fig. 5: Analysis of the first and second stages of the 151 Units of Information

When applying the questionnaire to the units of information, it was identified that some requirements contained units of consistent information, however in most cases they were incomplete, which led to inaccuracy. These data can be observed when considering the individual analysis of the questions of correctness and completeness. As illustrated in Figure 6(a), the analysis of the three

questions about consistency, the first question (The Units of Information or requirements must not be contradictory with each other nor with the domain) presented as a result that 103 Units of Information presented themselves without problems of contradiction to the domain while 48 had some kind of problem. The second question (Statements must be mutually exclusive) pointed out that 121 Units of Information were mutually exclusive, while 30 did not have this characteristic and, in turn, the third question (There should be no confrontation of terminology or terminology that causes doubts or misinterpretation) showed that 85 Units of Information did not have problems with doubts or misinterpretation related to terminology while 66 units had this type of problem.

Figure 6(b) shows that considering the first question related to completeness (All information referring to the requirement must be declared in the domain) obtained 89 units of information had their information declared in the domain while 62 did not. In the second question (The information should not contain objects, entities or indefinite terms), it was pointed out that 75 Units of Information had their information defined in the domain, while 76 presented some type of indefinite terms. Finally, the third question on completeness (No information should be missing) indicated that 52 Units of Information did not present an absence of information and 99 presented some type of lack of information.



(a) Result of Consistency issues (b) Result of Completeness issues

Fig. 6: Individual result of the evaluation questions of the Units of Information in the second phase of the method

5 Conclusion

When evaluating the 99 requirements, present in the 85 S.Os., it was observed that 36 were correct and of these, only 2 had some rework associated with them, i.e., 34 correct requirements have no rework. It can be observed that the method pointed out that 63 requirements presented some type of problem indicated by the the units of information analysis, indicating flaws of inconsistency or completeness leading to incorrectness. Thus, the 122 hours of rework were associated

with 63 requirements with some type of problem and only 2 requirements considered correct.

The application of the method proved to be feasible to assess the defined requirements. The application of the method makes it possible to assess the existence of problems during the registration and analysis of requirements, i.e,. problems associated with the process of requirements evolution. The information collected during the method applying can be useful for teams monitoring the process improvement, offering a valuable indicator of the before and the after of this practice.

It was observed that, unlike the development of a new software, in which the requirements activities present a greater demand of effort and concentration of the requirements engineers in the initial activities of the software development, the requirements evolution presents new demands in sparse periods. The method recommends, in its first phase, the description of the domain section, in which the requirement is inserted. This description can be positive for an analysis and a description of the requirements in the maintenance phases. In conclusion, the first phase of the method can also be used to conduct a better analysis of requirements.

References

1. Aguiar, L.G.F., da Silva, R.T., Genvigir, E.C., Fabri, J.A., L'Erário, A.: Um modelo para tratamento de ambiguidades em requisitos de evolução de sistemas jurídicos baseado em mapeamento conceitual. In: CIbSE. pp. 380–393 (2016)
2. Avdeenko, T., Pustovalova, N.: The ontology-based approach to support the completeness and consistency of the requirements specification. In: International Siberian Conference on Control and Communications (SIBCON). pp. 1–4. IEEE (2015)
3. Boehm, B.W.: Verifying and validating software requirements and design specifications. IEEE Software **1**(1), 75 (1984)
4. Chen, X., Zhang, W., Liang, P., He, K.: A replicated experiment on architecture pattern recommendation based on quality requirements. In: 5th International Conference on Software Engineering and Service Science (ICSESS). pp. 32–36. IEEE (2014)
5. Cordes, D., Carver, D.: Evaluation method for user requirements documents. Information and Software Technology **31**(4), 181–188 (1989)
6. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., et al.: Identifying and measuring quality in a software requirements specification. In: 1st International Software Metrics Symposium. pp. 141–152. IEEE (1993)
7. Ferrari, A., Spoletini, P., Gnesi, S.: Ambiguity as a resource to disclose tacit knowledge. In: 23rd International Requirements Engineering Conference (RE). pp. 26–35. IEEE (2015)
8. Ferrari, A., Spoletini, P., Gnesi, S.: Ambiguity cues in requirements elicitation interviews. In: 24th International Requirements Engineering Conference (RE). pp. 56–65. IEEE (2016)

9. Fockel, MarkusandHoltmann, J.: Reqpat: Efficientdocumentationof high-qualityrequirementsusingcontrolled natural language. In: IEEE 23rd InternationalRequirementsEngineeringConference (RE). pp. 280–281. IEEE (2015)
10. Génova, G., Fuentes, J.M., Llorens, J., Hurtado, O., Moreno, V.: A framework to measure and improve the quality of textual requirements. Requirements engineering **18**(1), 25–41 (2013)
11. Jani, H.M.: Applying case-based reasoning to software requirements specifications quality analysis system. In: The 2nd International Conference on Software Engineering and Data Mining. pp. 140–144. IEEE (2010)
12. Kamalrudin, M., Hosking, J., Grundy, J.: Improving requirements quality using essential use case interaction patterns. In: 33rd International Conference on Software Engineering (ICSE). pp. 531–540. IEEE (2011)
13. Muriana, L.M., Maciel, C., Mendes, F.F.: Checkingconsistencybetweendocumentsofrequirementsengineeringphase. In: XXXVIII Conferencia LatinoamericanaEnInformatica (CLEI). pp. 1–10. IEEE (2012)
14. Palomares, C., Franch, X., Quer, C.: Requirements reuse and patterns: a survey. In: International working conference on requirements engineering: foundation for software quality. pp. 301–308. Springer (2014)
15. Pekar, V., Felderer, M., Breu, R.: Improvement methods for software requirement specifications: a mapping study. In: 9th International Conference on the Quality of Information and Communications Technology (QUATIC). pp. 242–245. IEEE (2014)
16. Phillips, L.B., Aurum, A., Svensson, R.B.: Managing software quality requirements. In: 38th Euromicro Conference on Software Engineering and Advanced Applications (Euromicro DSD/SEAA). pp. 349–356. IEEE (2012)
17. Saito, S., Takeuchi, M., Hiraoka, M., Kitani, T., Aoyama, M.: Requirements clinic: Third party inspection methodology and practice for improving the quality of software requirements specifications. In: 21st IEEE International Requirements Engineering Conference (RE). pp. 290–295. IEEE (2013)
18. Sarmiento, E., Leite, J.C., Almentero, E., Alzamora, G.S.: Test scenario generation from natural language requirements descriptions based on petri-nets. Electronic Notes in Theoretical Computer Science **329**, 123–148 (2016)
19. Soares, H.A., Moura, R.S.: A methodology to guide writing software requirements specification document. In: Latin American Computing Conference (CLEI). pp. 1–11. IEEE (2015)
20. Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software technology **45**(14), 993–1009 (2003)
21. Zowghi, D., Gervasi, V.: The three cs of requirements: Consistency, completeness, and correctness. Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ) (04 2003)
22. Zowghi, D., Nurmuliani, N.: A study of the impact of requirements volatility on software project performance. In: Ninth Asia-Pacific Software Engineering Conference, 2002. pp. 3–11. IEEE (2002)