Quality Properties Evaluation for Software Requirements Specifications: An Exploratory Analysis

Roxana Saavedra[1], Luciana Ballejos[2], Mariel Ale[3]

[1]CIDISI – UTN – Facultad Regional Santa Fe – rsaavedra@frsf.utn.edu.ar
[2]CIDISI- UTN - Facultad Regional Santa Fe – lballejos@santafe-conicet.gov.ar
[3]CIDISI- UTN - Facultad Regional Santa Fe – male@frsf.utn.edu.ar

Abstract. Most software problems arise from deficiencies in the manner in which software requirements are elicited and expressed. Ensuring that the Software Requirements Specification document (SRS) has the necessary quality is crucial to the success of any software development project, since its information is used across all project stages. However, assessing the quality of a SRS is not a simple process, mainly by the multitude of proposals, often contradictory, of the attributes to be evaluated and the methodologies used for that purpose. This work is intended to be a compendium of the most important tendencies and strategies in the field that serves as a starting point for developing comprehensive models and tools for quality attributes evaluation in a SRS.

Keywords. Software Requirements Specification, quality attributes evaluation, quality models

1 Introduction

The primary measure for an information system to be successful is the degree in which it meets the intended purpose. Requirements Engineering (RE) is a subtask of Software Engineering which deals with the discovering of that purpose by identifying stakeholders and their needs, and documenting them for their future analysis, communication, and subsequent implementation [1].

In RE processes there is a continual need for efficiently managing the great volume of information and knowledge generated and used during all activities which compose the software development process. Thus, diverse are the challenges that must be considered when managing requirements-related information in software development projects. In this sense, ambiguous requirements must be minimized since they produce waste of time and repeated work.

Related to this, there exist in the literature diverse proposals in order to give guidance in the assessment of different attributes or properties for requirements, which helps in controlling if their specification is made in a correct way. Some of them also propose quality models to be considered when evaluating a Software Requirements Specification (SRS), the main deliverable produced in RE, which is used throughout the project [2,3].

The main goal of this paper is to analyze the state-of-the art in this area, in order to give a more consistent support for software engineers when generating requirements

specifications. Moreover, the results might be considered for the future development of tools for supporting automate or semi-automate evaluation of SRS quality.

The paper is organized as follows: Section 2 describes the research methodology used for the study. Section 3 presents diverse proposals in SRS quality properties, analyzing their similarities and differences. Meanwhile, Section 4 discusses and analyzes diverse proposals or approaches for evaluating the attributes described in the previous section. Related to this, Section 5 describes influences between these properties. Finally, Section 6 is devoted to discuss future trends in this area and the conclusions of the paper.

2 Research Methodology

The purpose of this study is to understand the trend of quality-related attributes assessment for SRS by examining published articles and offering, at the same time, insights and future directions in this area.

To this end, the following electronic journal databases were searched to provide an exhaustive bibliographic revision of research papers in the area:

- EBSCO Discovery Service,
- EconPapers,
- Compendex. Engineering Village,
- Engineering Village 2: Referex Engineering,
- National Digital Library of Theses and Dissertations (NDLTD),
- NTRS: NASA Technical Reports,
- Scopus. Elsevier,
- Social Science Research Network (SSRN),
- ACM Portal,
- IEEE Library.

The search process was performed based on three descriptors: “Software Requirements Specification”, “Quality Models” and “Quality Attributes Evaluation”. The selection of the descriptors was performed according to their different levels of generality, which allows refining the search results. The search process started from more general (e.g. "quality models") to more specific (e.g. "Software Requirements Specification") phrases, using the facilities of “search within the results” offered by most search engines. Despite the above, and due to the large quantity of papers returned by the databases, the following exclusion criteria were used:

- Given the limited number of indexed publications addressing this particular subject, the consideration of publications only in English. This was done in order to ensure that these articles have been accessed and validated by the widest possible audience. For the same reasons, non-refereed conference papers, unpublished master and doctoral dissertations were excluded.
- Because the idea of evaluating quality attributes for the SRS is relatively new, the search was restricted from 1990 onwards.
- Only research papers referring specifically to SRS quality attributes were chosen.

3 Quality Attributes

In general, quality attributes for a SRS are part of a quality model and should be used to assess the quality of a requirements document, or individual requirements contained in a SRS [4]. Thus, each attribute might be related to the entire SRS or to each requirement defined in it. This is due to, since for many attributes to be attained by the SRS, they have previously to be achieved for each requirement defined in it. Related to this, many of the authors proposing the same property coincide in their considerations, some issues regarding particularities in the analysis deserve to be pointed out. In the next subsections diverse definitions and proposals for each property are presented.

3.1 Unambiguous

A SRS is *unambiguous* if every requirement stated therein has only one possible interpretation [2,5,6]. Some authors give similar definitions for unambiguous requirements [7,8,9,10,11]. Pohl [7] states that a requirement is unambiguous if all stakeholders with approximately the same knowledge about the system and its context interpret the requirement in the same way. It also requires that, at least, each characteristic of the final product be described using a single unique term [5], and those terms that could have multiple meanings in a particular context, should be included in a glossary where its meaning becomes more specific [5,8].

3.2 Complete

The IEEE 830:2009 Standard [5] establishes that a SRS is *complete* if it contains the following elements: 1) all significant requirements; 2) definition of the responses of the software to all realizable classes of input data in all realizable classes of situations; 3) complete labels and references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure. Some authors also add the rule that all pages must be numbered and all referenced material must be present in SRS [2,12,13].

Furthermore, a SRS is *complete* if all relevant requirements are specified [2,7,8,10], and, therefore, irrelevant requirements should be absent [7,14]. Some authors [2,5], consider that a SRS is not complete if there are unfinished sections, i.e., there are marks "To Be Determined" (TBD). A requirement is *complete* if it is fully specified [7,6,8] and does not omit any piece of information that is relevant to some stakeholder [7].

3.3 Consistent

Some authors treat *internal* and *external consistency* as different quality properties [2,9], others seek only the *internal consistency* [5,7,14,15,16,17,18], and others consider the *internal* and *external consistency* as a same quality attribute [3,6,8,19].

3.3.1 Internally Consistent

A SRS is considered *internally consistent* if not subset of requirements outlined in it have conflicts [2,5,7,9,10,14]. Moreover, a requirement is *consistent* if the statements within it do not contradict each other [7].

3.3.2 Externally Consistent

A requirement in the SRS is considered *externally consistent* has no conflicts with any project documentation (i.e., system requirements specifications, statements of work, white papers, an early SRS version with which the new version must be compatible, and system requirements specifications of other systems to which this system must have interface). Thus, a SRS is considered externally consistent if the requirements described in it have no conflict with any project documentation [2,6,8,9,19].

IEEE 830:2009 [5] states that if a SRS disagrees with some superior-level document, such as a system requirements specification, then, this is not *correct* rather than *externally inconsistent* (see section 3.4).

3.4 Correct

A SRS is considered *correct* if every requirement is something required to build the system, i.e., each requirement contributes to the satisfaction of some need [2,5,6]. The SRS should be compared with any superior specification to ensure that it agrees. The customer or user can also determine whether the SRS correctly reflects the actual needs [5,8]. Instead, some authors define the quality attribute *validability/valid* when they establish that the client should be able to confirm that SRS requirements describe the system that meets their needs [10,14]. Pohl [7] states that a requirement is *correct* if the relevant stakeholders confirm that it is correct and require the system to perform the requirement completely. Thus, a requirement is *incorrect* if it unnecessary adds some functionality or quality property (gold platings).

3.5 Traceable

A SRS is considered *traceable* if each requirement has a clear source and is easily referenced in subsequent development phase or documentation [5,6]. IEEE 830:2009 [5] recommends two types of traceability: 1) *Backward traceability*, i.e., before the development stages. This is achieved when each requirement explicitly reference to its source in earlier documents. Some authors define this attribute as "*Traced*" [2,12,13,20]; 2) *Forward traceability*, i.e., all documents generated from the SRS. This is achieved when each requirement has a unique name or reference number. Some authors define it as "*Traceable*" [2,10,12,13,20]. A requirement is traceable if the origin and evolution as well as its impact and use in later stages of development is traceable [7,8].

Some authors treat separately *Traceable* and *Traced* quality properties [2,12,13, 20], others try these attributes together in Traceable [5,6,7,8], and others only consider Traceable attribute [10,14].

3.6 Verifiable (Testable)

A SRS is considered *verifiable* if every requirement stated therein can be verified [2,5,6]. A requirement is *verifiable* if there is a finite and cost-effective process with which a person or machine can check that the software product meets the requirement [2,3,5,6,7,8,14]. Pohl [7] states that if a requirement is underspecified, it is not objectively possible to decide if the requirement is realized as defined or not. It also states that, to facilitate verifiability, some acceptance criteria must be defined.

3.7 Modifiable

A SRS is considered *modifiable* if its structure and style are such that allow introducing easily, completely, and consistently any change, without affecting the structure and style [2,5,6,7,13]. To achieve modifiability a SRS must: 1) have a coherent and easy to use organization, a table of contents, an index, and explicit cross-references (see sections 3.11 and 3.14); 2) avoid redundancy (because problems can arise when a redundant requirement is altered in only one of the places where this occurs, resulting in a inconsistent SRS) (see section 3.12); 3) express each requirement separately (see section 3.13) [5,7,8].

3.8 Annotated by Relative Importance, Relative Stability or Version

A SRS is considered *ranked by importance* if each requirement in it has an identifier to indicate its importance [2,5,6,14]. One way to classify requirements is to distinguish classes of them as essential, conditional, and optional [2,5].

A SRS is considered *ranked by stability* if each requirement in it has an identifier to indicate the stability of that particular requirement [2,5,6,14]. Requirements stability can be expressed with the number of expected changes for any requirement [2,5].

A requirement is *rated* if its relevance has been determined and documented [7].

A SRS is considered *annotated by version* if a reader can easily determine which requirements will be satisfied in which program versions. One way of annotating requirements by version is to add a column in the SRS for each version of software to be produced and mark with an "X" next to each requirement in the respective columns [2].

Some authors distinguish the quality properties *Annotated by Relative Importance*, *Annotated by Relative Stability* and *Annotated by Version* [2,12,13]. Others consider *Annotated by Relative Importance* property as *Prioritized* [8], while others consider *Annotated by Relative Importance* and *Annotated by Relative Stability* as joint properties called *Ranked for Importance and/or Stability* or *Rated* [5,6,7,14].

3.9 Understandable

A SRS is considered *understandable* if its readers (customers, users, project managers, software developers, testers, and others) can easily comprehend the meaning of all requirements with a minimum of explanation [2,3,10].
According to Pohl [7], a requirement is *comprehensible* if its content is easy to understand. The comprehensibility of a requirement depends, among other things, on the selected document format and the stakeholders involved. Pohl [7] distinguishes between *Comprehensible* and *Readability*, while others define *Understandable* as part of *Readability* definition [2,3,10,20,21].

Furthermore, Pohl [7] defines that a SRS is *readable* if the reader can extract and easily understand its content. The readability and modifiability of a SRS is influenced by the structure and style of the document, so, in this sense, the SRS should have a coherent structure, each requirement should have a unique identification, redundancies should be avoided, and the requirements defined should be atomic. In order to reach these characteristics, diverse authors propose the use of IEEE Standard 830-2009 [5], which describes recommended practices for Software Requirements Specifications.

3.10 Concise

A SRS is considered *concise* if it is as short as possible without adversely affecting any other document quality. So, if there exist two SRS describing an identical system with identical quality measures, then the shorter the better [2,20].

3.11 Organized

A SRS is considered *organized* if its content is organized, that is, readers can easily locate information and logical relationships between adjacent sections are evident [2, 12,13,20]. According to Davis et al. [2], to achieve a useful organization: 1) a standard must be followed, and 2) one of the five organizational models must be used: group the functional requirements by user class, common stimulus, common response, feature or object. Some authors define the quality property *Organized* [2,12,13,20] and others only describe it as a required feature for achieving other quality attributes, such as *Modifiable* [5].

3.12 Not Redundant

A SRS is *redundant* if the same requirement is declared more than once [2,5,6,8]. Unlike other quality attributes, the redundancy is not necessarily bad. It is often used to increase readability of the SRS. However, it causes problems when a SRS is revised. If all occurrences of a redundant requirement are not modified, then the SRS becomes inconsistent [2,5,6,7,8]. If redundancy is necessary in the SRS, it should include explicit cross-references [5,7,8].

Some authors define the quality property *Not Redundant* [2,9]. Others treat this quality attribute as a guideline to be considered in requirements documentation [6], and others only describe it as a required feature for achieving other attribute quality as: *Modifiable* [5,7,8] and *Readability* (*Understandable*) [7].

3.13 Atomic

Each requirement in a SRS should be clearly determined and identified, without being mixed with other requirements [10]. A requirement is *atomic* if it describes a single and coherent event. A requirement is *not atomic* if it describes multiple isolated or just loosely coupled events which can be divided into several requirements [7].

Other authors treat this attribute similarly to the quality property *Not Redundant*, i.e., as a guideline to consider in requirements documentation [6] and as required feature to achieve the quality properties *Modifiable* [5,7,8] and *Readability* (*understandable*) [7].

3.14 Cross-Referenced

A SRS is considered *cross-referenced* if it cross-references are used to relate sections containing requirements to other sections containing: redundant requirements, more abstract or more detailed descriptions of the same requirements or requirements that depend on them or on which they depend [2].

3.15 Design Independent

A SRS is *design independent* if there are more than one system design and implementations that implement all requirements stated in it [2]. Génova et al. [10] call this quality property *Abstraction*, and state that requirements must tell what the system must do without telling how it must do it. Thus, a SRS should avoid excessive technical details about the implementation.

3.16 Electronically Stored

A SRS is considered *electronically stored* if the entire document has been produced with a word processor, was generated from a requirements database, or has been otherwise synthesized from some other form [2].

3.17 At Right Level of Abstraction/Detail

A SRS can provide different levels of detail [2,20]. A SRS being used as a contract between customer and developer should be relatively specific to ensure that the customer knows what is being acquired [2]. It is considered good practice to write the SRS requirements at a consistent level of detail [6].

3.18 Precise

The precision quality property is accomplished when all used terms in the SRS are concrete and well-defined [10]. Particularly, an SRS is considered *precise* if: 1) numerical quantities are used whenever possible, and 2) all numerical quantities have appropriate levels of precision [2].

3.19 Achievable

A SRS is *achievable* if there is at least one system that correctly implements all requirements stated in it [2]. Wiegers [8] calls this quality attribute *Feasible*, and defines it as the possibility to implement each requirement within the capabilities and limitations of the system and its operating environment.

3.20 Others Attributes

A SRS is considered *Prototypable* if there is a software tool capable of inputting it and providing a dynamic behavioral model that simulates the system behavior to build. This quality property is given only by Davis et al. [2].

A SRS is considered *Reusable* if it is possible to easily adopt or adapt their sentences, paragraphs and sections for use in a subsequent SRS [2]. This quality property is only described by Davis et al. [2].

A requirement is *Up to date* if it reflects the current status of the system and its context, such as the current stakeholders desire or current legal regulations [7]. It should be noted that this quality property was only proposed by Pohl [7].

4 Quality Properties Evaluation Analysis

There are many proposals for evaluating SRS quality attributes [2,3,6,10,11,12,13,14, 15,16,17,18,19,20,21]. The approaches found in the literature considering **vocabulary or language** include: *use of domain vocabulary*, related to the use of user vocabulary (glossary) in requirements descriptions [10,12,13]; *use of domain knowledge*, which considers interpretation and domain semantic knowledge (some authors use ontologies as knowledge resource) [2,15,16,17,18,19], and *natural language patterns detection* using keywords, key phrases and/or symbols as evidence of the occurrence for certain quality attributes [3,6,10,11,12,13,14,20,21].

Other approaches analyze **relations between requirements and artifacts** in SRS, in order to evaluate diverse attributes [10,15,16,17,18,19]. Among them, those proposing *overlapping between requirements* consider requirements referring to the same subject where contradictions between requirements, redundancy when there is a unnecessary repetition, and simple coupling when there is none of the above (and that implies some kind of dependency relationship) can be distinguished [10,15,16,17,18,19]. Also, other approach considers the evaluation of *requirements*

dependencies with other requirements or other artifacts of the development process [10]**.**

Diverse approaches are based in the analysis of **SRS structure** for evaluating some attributes [2,10,12,13,14,20,21]. Between them, *text structure in the SRS* considers requirements found in each hierarchical level of the SRS [10,14,20], while *specific characteristics of SRS document* includes presence of sections, table of contents and index, SRS size, etc. [2,12,13,14,20,21]. Meanwhile, *achievable features from SRS document* approach includes the evaluation of actual solution system designs, single system, etc. [2].

The analysis of the **requirements themselves** also is performed by diverse proposals [2,3,10,12,13,20]. In this sense, the *specific characteristics of a requirement* approach consider the use of explicit references, unique identifier, cross-references, versions and size for a requirement [2,3,10,12,13,20]. Moreover, the analysis of *deductible features from a requirement* includes the evaluation of cost and time required to verify a requirement [2].

Finally, as general approaches which consider the **jointly evaluation of groups of requirements**, approaches proposing *metrics that calculate the percentage of requirements that meet the analyzed attribute* [2], and *tools that use correct-by-construction paradigm* (where certain quality attributes are satisfied by the mere fact that tool be used to generate the SRS) are included in the analysis [12,13].

Thus, while Table 1 resumes which references have a concrete proposal on the evaluation or assessment techniques for each attribute, the approaches found together with the attributes evaluated by them are summarized afterwards.

The *use of domain vocabulary* is proposed for the evaluation of Unambiguous and Understandable [12,13], Atomic and Precise [10]. On the other hand, The *use of domain knowledge*, is included in assessment techniques proposed for Unambiguous [2,15,16], Complete [2,15,16,17,18,19], Internally Consistent [2], Correct [15,16,17,18], Understandable [2] and Atomic [18] quality attributes.

Moreover, the *use of natural language patterns* is proposed for evaluation techniques related to Unambiguous [11,13,14,20,21], Complete [3,12,13,20,21], Verifiable [3,21], Annotated by Relative Importance and Relative Stability [13], Understandable [3,10,14,20,21], Atomic [6,10], Design Independent and Precise [10] quality attributes.

Furthermore, within the *use of overlap between requirements* approach, proposals exist for the evaluation of Internally Consistent [15,16,17,18,19], Not Redundant [17], Unambiguous, Traceable, Understandable and Atomic [10] quality attributes.

Within the *use of requirements dependencies with other requirements or other artifacts of the development process* approach, evaluation techniques are proposed for quality attributes such as Traceable, Understandable and Atomic [10].

Also, the *use of text structure in the SRS* approach is considered when proposing evaluation techniques for quality attributes such as Understandable (considering the degree of nesting between requirements) [10], Organized (number of requirements at each hierarchical level) and Right level of Abstraction/Detail (considers the specification depth, i.e., the number of imperatives in each level of the SRS text structure) [14,20].

Table 1. References with Concrete Proposal on Attribute Evaluation Techniques

Reference / Property	[2]	[3]	[6]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
Achievable	x														
Annotated by Relative Importance, Relative Stability or Version	x						x								
Atomic			x	x								x			
Complete	x	x				x	x		x	x	x	x	x	x	x
Concise	x							x						x	x
Correct	x			x					x	x	x	x			
Cross-Referenced	x														
Design Independent	x			x											
Electronically Stored	x						x								
Externally Consistent	x														
Internally Consistent	x	x							x	x	x	x	x		
Modifiable	x						x								
Not Redundant	x										x				
Organized	x					x	x	x						x	
Precise				x											
Prototypable	x														
Reusable	x														
Right Level of Abstraction/Detail	x							x						x	
Traceable	x			x		x	x							x	
Traced	x					x	x								
Unambiguous	x			x	x	x	x	x	x	x				x	x
Understandable	x	x		x		x	x	x						x	x
Verifiable (Testable)	x	x		x											x

The *use of specific characteristics of the SRS document* approach, is applied in assessment techniques related to quality attributes like Complete (considering that certain sections are present in the SRS) and Organized (considering whether the required sections are present in the required order and with the content required) [12,13], Concise (considering the size of the SRS) [2,14,20,21], Modifiable (considering the presence of table of contents and index, and the degree of cohesion and coupling of SRS sections), Electronically Stored (considers SRS volume that has been electronically stored) and Reusable (considers paragraphs in the SRS that exhibit reuse properties) [2].

Moreover, within the *achievable features from SRS document* approach, a set of evaluation techniques for diverse quality attributes is proposed. The attributes are: Design Independent (number of actual solution system designs that satisfy all requirements of SRS), Achievable (considering the existence of a single system), Prototypable (considers whether SRS can be partially written in a executable, interpretable or prototypable language) Reusable (considering whether the content of the SRS has been used in subsequent SRS) [2].

Also, the *use of specific characteristics of a requirement* approach is used for evaluating quality attributes like Internally Consistent (considering the use of explicit references in a requirement) [3], Traceable (considering using requirement unique identifier) [2,12,13,20], Traced (considering the use of cross-references) [2], Correct and Verifiable (considering the number of versions of a requirement) [10], Atomic (considering the size of a requirement) [10].

Moreover, in the *use of requirement deductible characteristics* approach, Davis et al. [2] propose assessment techniques for the quality attribute Verifiable (considering the cost and time required to verify the requirement).

In the *using metrics that calculate the percentage of requirements that meet the attribute in question* approach, Davis et al. [2] propose diverse metrics for evaluating quality attributes such as Externally Consistent, Correct, Annotated by Relative Importance, Relative Stability and Version, Not Redundant. The metrics proposed, in general, consider the ratio between the requirements satisfying some specific attribute, over the total number of requirements in the SRS. However, no concrete proposals are described by the authors, in order to give guidance over how considering or identifying the requirements which satisfy each attribute.

In the case of the *use of tools in the correct-by-construction paradigm* approach, evaluation techniques are proposed for Traced [12,13], Modifiable, Annotated by Version and Electronically Stored [13] quality attributes.

Finally, Davis et al. [2] indicate that Organized, Cross-Referenced, and Right Level of Abstraction/Detail attributes cannot be measured for different reasons. The authors affirm that "organization" is purely subjective and, thus, it cannot be measured. On the other hand, Cross-Referenced cannot be measured because there is no way to determine how many cross-references are appropriate in an SRS. In relation to Right Level of Abstraction/Detail, measuring the appropriateness of the SRS level of abstraction is highly scenario-dependent. Moreover, for Up to Date attribute was not found in the literature any evaluation proposal.

5 Influences between Quality Attributes

The possibility of creating a SRS of reasonable quality exists, but it has to be considered that most of the quality properties mentioned in this paper have positive or negative effect on other properties (see Figure 1). Because of this, it is necessary to determine which quality attributes are most important to the project, in order to achieve them. Below, the effects between attributes found in the literature analyzed are detailed:

- Generally, requirements considered unverifiable are ambiguous [5,8,10], incomplete or inconsistent [8,10], or infeasible requirements [8].
- The elimination of ambiguity in the SRS requires adding formality, which is not understood by people who are not computer experts, for example, users or customers [2,8,10].
- The less ambiguous is the SRS, the easier its modification [10].
- If a SRS is not complete in terms of objectives, rules, facts, and constraints of the problem domain, then it is considered incorrect for that domain [9,10].
- Exceeding the completeness of the SRS causes losing conciseness [2].

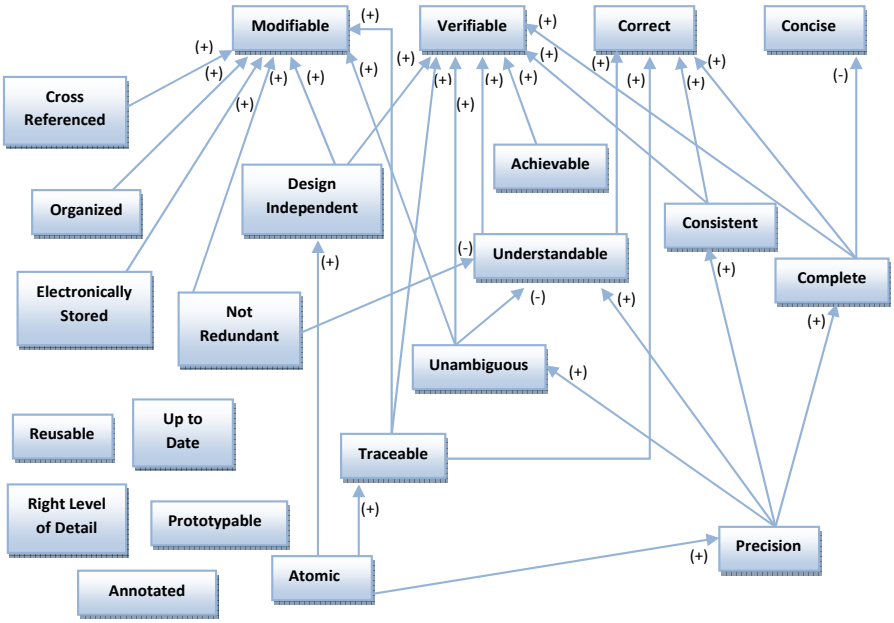

Fig. 1. Influence between quality properties.

- It is not always possible to significantly reduce the SRS size without negatively affecting other quality attributes [2].
- If a SRS is inconsistent, then the client cannot confirm that requirements effectively express the system that responds to their needs, i.e., its correctness [10].
- A SRS is more modifiable if it is traceable, organized, cross-referenced and stored electronically [2].
- Traceability facilitates correctness [5] and verifiability [10].
- Requirements not understood are not verifiable and its correctness may not be validated [10].
- The redundancy is often used to increase readability of the SRS. Furthermore, redundancy can negatively affect the modifiability, since not changing all occurrences of a redundant requirement generates an inconsistent SRS [2].
- A non-atomic requirement has the risk of excessive detail, and thus, may affect design independence [10].
- Atomicity influences requirements precision and traceability [10].
- Design independence can help verifiability. The presence of requirements technical details is more difficult to verify and modify [10].
- A more precise language helps to write more complete, consistent, understandable and unambiguous requirements [10].

Figure 1 resumes the influences among quality attributes previously described. In addition, it can be noted that no dependencies can be found for some attributes, due to

it is possible to reach them without affecting other attributes. They are: Reusable, Up to Date, Annotated, Prototypable and At Right Level of Detail.

6 Conclusions

In this work, an exploratory analysis of various proposals for quality attributes evaluation of a SRS was performed. On the one hand, there exist different quality models for SRS. In the ones analyzed, some characteristics are worth to note. Firstly, some of the attributes are used by most models, for example, unambiguous and complete. Secondly, there are attributes which are considered separately by some authors, while others use them together, aggregated in a single attribute, e.g., Traced and Traceable. Additionally, other attributes, although being the same, are differently referenced by some authors: for example, Independent Design is also called Abstraction. Finally, some attributes are only used by the author who defines it, for example, Up to Date. Furthermore, it is clear that some attributes are applicable to the entire SRS, while others apply only to individual requirements.

On the other hand, there are many proposals describing approaches for SRS quality attributes evaluation. Some of them are only conceptual and, therefore, difficult to automate, since they require a high degree of human reviewers intervention. Other proposals describe tools implementations in different technologies that pose heuristics or metrics for just some quality attributes. Usually, this assessment only can be achieved with the technology used and, in many cases, does not cover all possible cases or have certain shortcomings that require optimization.

To conclude, is important to note that the definition of heuristics, metrics and/or objectively measurable indicators is needed in order to cover all possible cases for requirements and SRS quality attributes evaluation in a software project. It is also required the construction of a supporting tool for implementing these defined heuristics, metrics and indicators, so an automatically assessing can be performed for obtaining quality SRS. This constitutes the main goal for future research in this area.

7 References

1. Nuseibeh, B.; Easterbrook, S.: Requirements engineering: a roadmap. In: Proc. Conference on The Future of Software Engineering, pp. 35-46. (2000).
2. Davis, A.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A.; Kincaid, G.; Ledeboer, G.; Reynolds, P.; Sitaram, P.; Ta, A.; Theofanos, M.: Identifying and measuring quality in a software requirements specification. In: Proc. 1st International Software Metrics Symposium, pp. 141-152. (1993).
3. Fabbrini, F.; Fusani, M.; Gnesi, S.; Lami, G.: An Automatic Quality Evaluation for Natural Language Requirements. In: Proc. 7th International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland. (2001).
4. Gnesi, S.: Analysis of Software Requirements. IEI-CNR Pisa http://www.iei.pi.cnr.it/ERI/iei/qmslideseri.ppt (2000).
5. IEEE Recommended Practice for Software Requirements Specifications. IEEE Standard 830-1998 (R2009), Institute of Electrical and Electronics Engineers. (2009).

6. Swathi, G.; Jagan, A.; Prasad, Ch: Writing Software Requirements Specification Quality Requirements: An Approach to Manage Requirements Volatility. Int. J. Comp. Tech. Appl., **2**(3), 631-638. (2011).
7. Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer-Verlag Berlin Heidelberg. (2010).
8. Wiegers, K.: Software Requirements, Second Edition. Microsoft Press. (2003).
9. Loucopoulos, P.; Karakostas, V.: System Requirements Engineering. McGraw-Hill, Inc. New York, NY, USA. (1995).
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), pp 25-41 (2013).
11. Tjong, S.F.: Avoiding Ambiguity in Requirements Specifications. PhD Thesis. University of Nottingham Malaysia Campus, Faculty of Engineering & Computer Science, Malaysia. (2008).
12. Durán, A.; Bernárdez, B.; Ruiz, A.; Toro, M.: An XML–based Approach for the Automatic Verification of Software Requirements Specifications. In: Proc. 4th Workshop on Requirements Engineering, pp. 181-194. (2001).
13. Durán, A.; Ruiz-Cortés, A.; Corchuelo, R.; Toro, M.: Supporting requirements verification using XSLT. In: Proc. IEEE Joint International Conference on Requirements Engineering, pp. 165–172. (2002).
14. Wilson, W.M.: Writing Effective Requirements Specifications. Software Technology Conference Proceedings. (1997).
15. Kaiya, H.; Saeki, M.: Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In: Proc. 5th International Conference on Quality Software, pp. 223-230. (2005).
16. Kaiya, H.; Saeki, M.: Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In: Proc. 14th IEEE International Requirements Engineering Conference, pp. 189-198. (2006).
17. Dzung, D.; Ohnishi, A.: Ontology-based Reasoning in Requirements Elicitation. 7th IEEE Int. Conf. on Software Engineering and Formal Methods, pp. 263-272. (2009).
18. Hu, H.; Zhang, L.: Ye, C.: Semantic-based Requirements Analysis and Verification. In: International Conference on Electronics and Information Engineering, pp. 241-246. (2010).
19. Verma, K.; Kass, A.: Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents. In: Proc. 7th International Conference on The Semantic Web, pp. 751–763. (2008).
20. Ali, M.J.: Metrics for Requirements Engineering. Master's Thesis. Umea Univ. (2006).
21. Rosenberg, L.; Hammer, T.; Huffman, L.: Requirements, testing, and metrics. In: 16th Pacific Northwest Software Quality Conference, Utah, USA. (1998).