A Framework to Measure and Improve the Quality of Textual Requirements

Article *in* Requirements Engineering · March 2011

DOI: 10.1007/s00766-011-0134-z

CITATIONS
58

READS
712

5 authors, including:

Gonzalo Génova
Universidad Carlos III de Madrid, Leganés, Spain
63 PUBLICATIONS **417** CITATIONS
SEE PROFILE

José M. Fuentes
The REUSE Company
19 PUBLICATIONS **148** CITATIONS
SEE PROFILE

Juan Llorens
University Carlos III de Madrid
128 PUBLICATIONS **649** CITATIONS
SEE PROFILE

Omar Hurtado
Universidad de Piura
11 PUBLICATIONS **90** CITATIONS
SEE PROFILE

A Framework to Measure and Improve the Quality of Textual Requirements

Gonzalo Génova[1*], José M. Fuentes[2], Juan Llorens[1], Omar Hurtado[1], Valentín Moreno[1]

[1]{ggenova, llorens, ohurtado, vmpelayo}@inf.uc3m.es

Knowledge Reuse Group, Universidad Carlos III de Madrid

Avda. Universidad 30, 28911 Leganés (Madrid), Spain

[2] josemiguel.fuentes@reusecompany.com

The Reuse Company

C. Margarita Salas, 16 (Parque Tecnológico Legatec), 28919 Leganés (Madrid), Spain

* Corresponding author: ggenova@inf.uc3m.es, tel +34916249107, fax +34916249129

Abstract. Improving the quality of software demands quality controls since the very beginning of the development process, i.e. requirements capture and writing. Automating quality metrics may entail considerable savings, as opposed to tedious, manually performed evaluations. We present some indicators for measuring quality in textual requirements, as well as a tool that computes quality measures in a fully automated way. We want to emphasize that the final goal must be *measure to improve.* Reducing quality management to the acquisition of a numerical evaluation would crash against the strong opposition of requirements engineers themselves, who would not see in the measurement process the aid of a counselor, but a policeman mechanism of penalties. To avoid this, quality indicators must first of all point out concrete defects and provide suggestions for improvement. The final result will not only be an improvement in the quality of requirements, but also an improvement in the writing skills of requirements engineers.

Keywords: *Requirements Quality, Quality Properties, Quality Indicators, Quality Metrics, Quality Measurement Tool, Natural Language Processing.*

1. Introduction

Requirements engineering is the branch of software engineering that deals with one of the first stages in the software development process: understanding client needs and defining them in the form of a structured set of requirements an information system must meet. In this process we can distinguish two different phases [4]: a) *requirements capture*, i.e. the careful interaction with all those interested in the software application, usually called *stakeholders*, aimed at the fundamental purpose of acquiring information, the *raw requirements*; and b) *requirements analysis*, i.e. the methodical study of the acquired information to yield a true understanding of what the system must do to satisfy client needs, so that through a process of formalization, refinement and structuring, efficiently managed, requirements reach a concrete and detailed expression, the *refined requirements*. This process can be summarized in the words of Loucopoulos and Karakostas, who define requirements engineering as the "systematic process of developing requirements through an iterative co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained" [37].

As it has been often pointed out [4, 7, 22, 46, 47], most of the defects in the delivered software originate in a deficient requirements analysis, and they are generally the most difficult ones to repair. Therefore, it is of major importance to provide this field with engineering discipline, particularly by means of *quality controls since the very beginning of the process* [54]. If we do not demand that the requirements meet certain quality criteria, then it will be more difficult to search for quality in later development phases. Although there is no complete agreement on the amount of effort needed in the requirements phase of software projects (estimates between 5% and 16% of total effort, with too much variance), empirical studies show that rework to locating and correcting defects found during testing is very significant (around 40-50%); thus, more investment upfront will lead to less rework [8, 43]. Obtaining the right requirements is a difficult and iterative process, in which engineers must respond to the double challenge of *discovering* and *formalizing* the wants and needs that the clients usually are able to describe only in a confused, incomplete and messy way [10]. The success of this process requires fluid collaboration and communication between clients and software engineers: the more complete and unambiguous is the set of requirements, the

more likely the success of the project will be [30]. This need of communication among all stakeholders has been the cause that the privileged form to express requirements is *natural language* [27, 28, 52], as opposed to formal languages that are more or less inaccessible, mainly to clients. The majority of the documents available for requirements analysis are provided by the client and couched in *real* natural language, therefore the use of linguistic techniques and tools may perform a crucial role in providing support for requirements analysis [34]. The market has ruled also in this sense, since nearly all requirements management tools are natural language oriented [36].

Once we have assumed that the most common form to express requirements is natural language, there are some possibilities with regard to format: we can find *textual documents* not explicitly structured, where domain descriptions are mixed up with specifications of the system-to-build, and where requirements do not even appear individualized; on the opposite end, we find *specialized representations* (for example, by means of data bases), where requirements, their constitutive elements, and the relationships among them, are perfectly identified; we can find also many intermediate stages, such as the use of *predefined templates* for requirements in well structured textual documents. We should not disdain the use of general-purpose word processors or spreadsheets to express requirements, basically due to their flexibility and the nowadays-generalized knowledge about this kind of tools, which avoids the acquisition and learning of specific tools. However, this very flexibility may cause a lack of rigor in the treatment of requirements [32], and no doubt makes some aspects of requirements management more difficult, such as version control, use of a normalized vocabulary, traceability relationships with other requirements and software artefacts, multi-user editing of requirements, etc. In particular, it makes difficult to obtain quality metrics. Summing up, it is necessary to find the balance between a comfortable and flexible tool, and one that achieves the maximum of requirements management automation.

In this paper we present a framework to obtain low-level quality indicators in textual requirements, such as requirement size, number of ambiguous terms, imperative verbal forms, overlapping requirements, and so on. This framework has been implemented in a tool that computes quality metrics in a fully automated way, *RQA Requirements Quality Analyzer* [48]. The tool user has to define a set of parameters to adjust the tool to the desired quality levels; however, the *tuning* of parameters (for example against a benchmark of previously evaluated requirements) is out of the scope of this work. This tool follows the trend and integrates many ideas of other tools developed in the past, particularly ARM [35, 52] and QuARS [13, 49]. Every quality measurement has an added cost in time and money, so that automating quality metrics may entail considerable savings, as opposed to costly, manually performed evaluations (so costly that they rarely become real). Now, it is important to emphasize that the final goal must be *measure to improve*. Reducing quality management to the acquisition of a numerical evaluation would crash against the strong opposition of requirements engineers themselves, who would not see in the measurement process the aid of a counselor, but a policeman mechanism of penalties. To avoid this, quality indicators must not provide numerical evaluations only, but first of all they must point out *concrete defects* and provide *suggestions for improvement*, just like a spell and grammar checker can help to improve the quality of a text. In this sense, our work focuses on the analysis of textual quality of requirements, using formal requirements documents as input data; we are not directly concerned with the meaning or content of requirements, nor with the process to elicit the right requirements, i.e. what the stakeholders really want. Computing low-level metrics can be an aid for *writing the requirements right*, but not for *writing the right requirements* [6, 16]. Of course, low-level metrics cannot replace expert judgment on the high-level quality of requirements, but they can provide valuable hints to improve them. Our purpose has been to propose a helpful solution, a tool that can be used by practitioners (requirements engineers, quality managers, etc.) because it abstracts away the complexity of natural language processing, and gives them practical, understandable improvement suggestions.

We have followed a *design science approach*, i.e. a scientific method that is not aimed at the study and analysis of the truth of cause-effect relationships in natural phenomena (the classical scientific method), but at the utility of means-ends relations in the design and construction of human artifacts [53]. Of course, design sciences are grounded on natural sciences: valid cause-effect relationships give the possibility to build useful means-ends relations. The design of human artifacts such as computer tools is a sophisticated task that, if performed with a specific and rigorous research method, contributes to the development of scientific knowledge [19]. The method usually starts with the identification of a need (in this case, a demand for better textual quality in requirements), followed by artifact design, construction and evaluation [20, 51].

According to this method, the rest of the paper is structured as follows. In Section 2 we systematize and justify the list of desirable properties we expect a requirements specification will

meet, i.e. the first step in the method: identification of the need. The next three Sections constitute the theoretical framework that serves as a basis for the design and construction of a practical tool: in Section 3 we explain how to define quantifiable indicators from direct measurements on requirements, by means of step transformation functions; in Section 4 we present a taxonomy of indicators and their relationships with the desirable properties; in Section 5 we deal with the aggregation of individual measures to get the quality index of a requirement and the global quality measure of a set of requirements. The evaluation step follows in Section 6, where we present the quality management tool we have developed, we compare it with other available requirements quality tools (based on an external survey by the RAMP project [14, 15]), and we report about its reception by industry. Finally, Section 7 contains brief conclusive reflections.

2. Desirable properties of requirements

In this Section we answer the following question: *What should we measure?* In other words, we define what we understand as good or bad quality of a requirement or set of requirements. Many authors and methodologies distinguish between *qualitative desirable properties* of requirements, dependent on subjective judgment, and *quantitative measurable indicators*, based on objectivable characteristics of requirements [16, 42, 52]. The different lists of desirable properties found in the literature are complementary: completeness, correctness, hierarchization, unambiguity, consistency, modifiability, traceability, verifiability, validability, testability [24, 25, 42, 52]; completeness, consistency, clarity, testability, existence of error conditions, traceability [4, 26]; feasibility, verifiability, understandability, traceability, adaptability [44]; feasibility, clarity, traceability, testability, consistency, abstraction, unambiguity, quantification, structuration, existence of source [39]; unambiguity, completeness, consistency, understandability [13]. Being something different, quantitative indicators (defined and explained in the next Section) are more or less directly related to qualitative properties, so that the indicators can be used as an objective measure of subjective quality.

We have synthesized the different lists of desirable properties as follows (see Figure 1 for a graphical arrangement of the properties according to their mutual influence):

- *Validability*: the client must be able to confirm (validate) that the requirements effectively express the system that answers his or her needs.
- *Verifiability*: the engineer must be able to check (verify) that the produced system meets the specified system.
- *Modifiability*: requirements must be modifiable to ease system modification for maintenance.
- *Completeness*: all needs are covered by the set of requirements.
- *Consistency*: there are no contradictions among requirements.
- *Understandability*: the requirements are correctly understood without difficulty.
- *Unambiguity*: there exists only one interpretation for each requirement (unambiguity and understandability are interrelated, they could be even the same property; some authors have coined the term ‘uniguity’ [3, 38], but we prefer the more traditional form, even if it is a bit awkward).
- *Traceability*: there exists an explicit relationship of each requirement with design, implementation and testing artefacts.
- *Abstraction*: the requirements tell what the application must do without telling how it must do it, i.e. excess of technical detail about the implementation must be avoided in the specification of the requirements.
- *Precision*: all used terms are concrete and well defined.
- *Atomicity*: each requirement is clearly determined and identified, without mixing it with other requirements.

Note that we are following here Barry Boehm’s definition of validation and verification [5]:

- Validation: To establish the fitness or worth of a software product for its operational mission (from the Latin *valere*, to be worth). Informally, are we building the right product?
- Verification: To establish the truth of correspondence between a software product and its specification (from the Latin *veritas*, truth). Informally, are we building the product right?

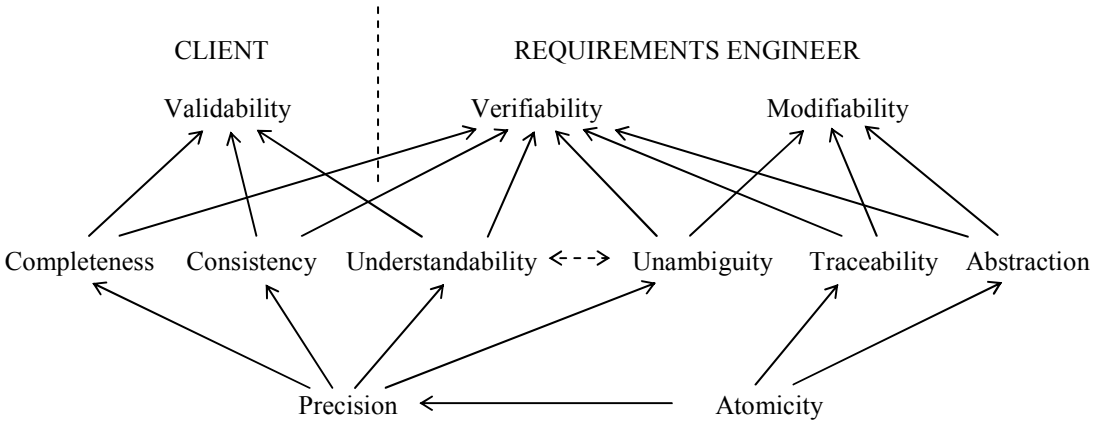

Fig. 1 Influence of qualitative properties among them

Our starting point to justify the list of desirable properties a good requirements specification should meet is the following question: *What is the goal of requirements?* The answer will show us what we should consider as good or bad requirements. Now then, the goal of requirements is to *specify a software system*: the system the client needs, and the system the engineer produces and maintains.

From the point of view of the client, then, the final quality of requirements (individually and as a set) is *validability*, i.e., the client must be able to confirm that the requirements effectively express the system that answers his or her needs. This property can be further unfolded, in a second level, into three other properties: *completeness*, *consistency* and *understandability*.

From the point of view of the requirements engineer, instead, we can say that the two essential qualities are: *verifiability*, i.e. it is possible to check that the produced system meets the specified system; and *modifiability*, since, if it is difficult to modify the requirements, then the produced system will be equally difficult to modify during the maintenance phase of the lifecycle. Verifiability depends on the same three second order properties we have mentioned before (completeness, consistency and understandability), as well as on three other properties: *unambiguity*, *traceability* and *abstraction*; modifiability depends also closely on these three last properties. Unambiguity and understandability are interrelated (according to some, they would be even the same property), since, if a requirement is ambiguous, then it cannot be properly understood. Note that abstraction is related with the lack of technical details about the implementation; the presence of non-technical details is not against abstraction as we have defined it. Therefore, abstraction and unambiguity are not opposing concepts: a requirement must have a single interpretation at the abstract level of specification (the 'what') and must avoid technical details about the implementation (the 'how'); having several possible implementations does not make the requirement ambiguous from the abstract point of view. The presence of non-technical details certainly can improve unambiguity and help verifiability and modifiability; on the contrary, the presence of technical details (lack of abstraction) makes the requirements excessively dependent on the implementation, thus more difficult to verify and modify.

We can finally distinguish a third more basic level in this taxonomy of desirable properties, which would contain two additional, even more elementary properties: *precision* and *atomicity*. Completeness, consistency, understandability and unambiguity depend on precision: a more precise language helps to write requirements that are more complete, consistent, understandable and unambiguous. Atomicity, on the other hand, influences precision, abstraction (a non-atomic requirement risks to have excessive detail), and especially traceability, and hence modifiability during the maintenance phase of the lifecycle. Indeed, if requirements constitute a monolithic system, instead of a structured system composed of individual elements, well identified and related to each other, then it will not be easy neither to modify the requirements nor the produced system.

3. Measurable indicators of requirements

In the previous Section we have reflected on *what we should measure*. In this Section we address *how to measure* those desirable properties we have identified. We have already mentioned that these properties depend on subjective judgment, which does not mean they are arbitrary, but that they are not easy to quantify. Therefore we need to define a series of quantifiable indicators that are related with the qualitative properties we wish to evaluate. For example, we can use as an

indicator the *size* of a requirement, measured by the number of words in its description. Size affects the properties of the requirement, particularly atomicity, and all the others through atomicity.

Which size indicates a good atomicity? Size, taken as a direct numerical measure of a certain text, does not indicate anything by itself: obviously, a requirement is not better by the simple fact of being bigger or smaller. Therefore, measures are generally classified into a set of discrete levels, usually three, or a similar number: High, Medium, Low; Good, Medium, Bad; etc. In the case of the size of the requirement, it seems clear that a requirement must be neither very short nor very long. In other words, in order to transform a primitive measure such as the *numerical size* into an indicator of the *adequate size* we need a convex step function (i.e., mountain shaped, increasing-decreasing), like the one in Figure 2. The size is considered Good if it falls between the limits x_2 and x_3, Bad if it is lower than x_1 or higher than x_4, and Medium within the intervals x_1-x_2 and x_3-x_4. The difficulty consists, precisely, in determining the values of the function parameters (the interval ends).

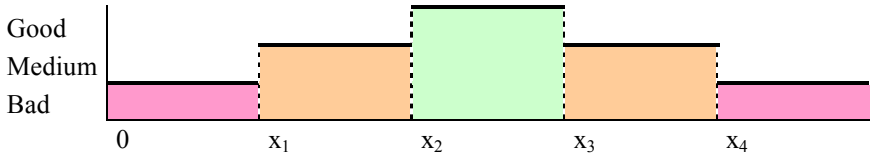

Fig. 2 The *adequate size* indicator as a function of the *numerical size* (increasing-decreasing step function)

In the same way as with size, we can proceed with other indicators such as readability index, number of imperative verbal forms, number of ambiguous terms, number of domain terms, etc. Each one of these indicators (which we will see later in detail) has different characteristics, so that the *step transformation function* will be different for each one. That is, for some indicators, such as size, an intermediate value of the primitive numerical measure is considered adequate, whilst for others an extreme value is adequate. For example, having no ambiguous terms at all is most adequate, and the more there are the worse the result of the indicator: it would be a decreasing step function.

Generally, we can distinguish four basic kinds of step functions: *increasing, decreasing, convex* and *concave* (see Figure 3). The two first ones require two parameters to be fully defined (x_1, x_2), whilst the two last ones (mountain, increasing-decreasing and valley, decreasing-increasing) require four parameters (x_1, x_2, x_3, x_4). To be precise, we define the intervals as closed in the lower end and open in the higher end, i.e. $[0, x_1)$, $[x_1, x_2)$, $[x_2, \infty)$, or else $[0, x_1)$, $[x_1, x_2)$, $[x_2, x_3)$, $[x_3, x_4)$, $[x_4, \infty)$. Thus, the increasing step function yields a Medium value if the numerical count x meets the double condition $(x \geq x_1)$ and $(x < x_2)$, and analogously in the remaining cases. Examples:

- Requirement size in words: convex step function, intervals defined for 0-5-10-50-100-∞ (having between 10 and 49 words is considered optimal).
- Number of ambiguous terms: decreasing step function, intervals defined for 0-1-3-∞ (having no ambiguous terms at all is considered optimal).

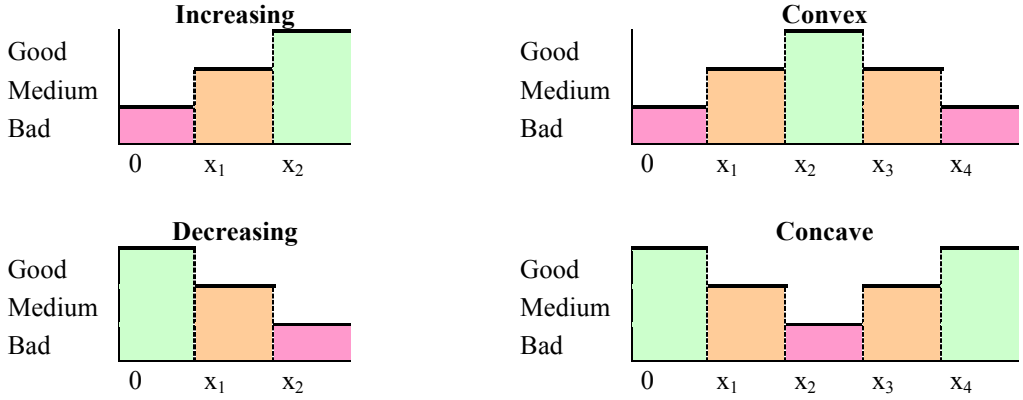

Fig. 3 Four basic kinds of step functions to transform numerical measures into quality indicators

If we want to distinguish only two quality levels (Good, Bad) in a certain case, then it will be enough to equate the parameters $x_1 = x_2$ and $x_3 = x_4$ in the definition of the step function. In our

experience, the two most frequent kinds of step functions are decreasing and convex, but we define the other two kinds for the sake of genericity. We do not think it is necessary, for such elementary measures as these, to distinguish more kinds of step functions (i.e. with more inflection points), such as increasing-decreasing-increasing, etc. Therefore, these four basic kinds would be enough to transform any primitive numerical measure we can get from the requirements into a quality indicator.

Step functions, besides serving for the purpose of transforming a numerical measure into a quality indicator, are also useful to *regularize heterogeneous numerical measures*. For example, the numerical range of a requirement size measured in words may fall between a few units and a few hundreds, whilst the range of ambiguous terms would usually be of a few units; some measures will be integer, whilst other will be decimal; etc.

4. Taxonomy of indicators

All we have said up to now would be scarcely useful if we are not clear as to which indicators are meaningful with regard to requirements quality. There are some pioneering works in this field, in particular NASA's Automated Requirement Measurement (ARM) software [35, 52], and plenty of literature in more recent years [1, 13, 27, 28, 30, 42], with similar applications in the analysis of textual scenarios for use cases [16]. Next we present some of the most frequently used indicators, classifying them in four main categories:

- *Morphological indicators*, such as size, which measure text properties from a purely formal point of view, without considering contents at all.
- *Lexical indicators*, such as number of ambiguous terms, which measure properties relative to text contents and require some kind of reference information (in this case, the list of user-defined ambiguous terms).
- *Analytical indicators*, such as usage of verbal forms, which require a textual analysis of requirements by means of relatively complex linguistic tools.
- *Relational indicators*, such as number of overlappings with other requirements, which measure structural properties of the set of requirements, rather than properties of individual requirements.

Note that this categorization is based on the measurement procedure, which generally involves a growing computational complexity; we could equally classify indicators according to the desirable property they are related to (atomicity, precision, unambiguity, traceability, etc.), or according to the shape of their step transformation function, etc.

4.1. Morphological indicators

The two easiest measures to get from a requirement are size and readability index. We consider also the count of punctuation signs, acronyms and abbreviations.

4.1.1. Size. The *size* of a requirement can be measured in characters, in words (the most intuitive), in sentences, or in paragraphs. Note that the number of paragraphs will scarcely discriminate, since it will always be rather small; besides, we must be careful on counting paragraphs, since if the requirement contains enumerated lists then the number of paragraphs may result misleadingly too big. As we have already mentioned, the adequate size of a requirement will be neither too big nor too small, therefore it will have a convex step function. The size of the requirement directly affects the desirable property of atomicity, and all the others through this one, particularly traceability, verifiability and modifiability. Note that computing the size requires that the requirement be explicitly identified. This may seem obvious nowadays, but a few years ago it was not always so; instead, you could often find requirements specifications consisting of a continuous text where requirements were not perfectly determined and enumerated [52].

4.1.2. Readability. *Readability indexes* try to measure the degree of difficulty to read a text. Typographical legibility must not be confused with linguistic readability, which is the one we are interested in. These indexes, often incorporated in regular word processors, are based on *average syllables per word* (or characters per word, a lot easier to measure) and *average words per sentence*. The underlying idea is that a text is more readable when average sentences and words are shorter. For example, the Flesch readability index [18] follows the formula R_{Flesch} = 206.835 - (1.015 x WPS) - (84.6 x SPW), where WPS and SPW stand respectively for average words per sentence and average syllables per word. The result evaluates the text in a range of 100 points; the higher the result, the easier to understand the document, therefore the corresponding step function is increasing. Obviously, the coefficients in these formulae must be adapted to the peculiarities of each language. In the case of Spanish, José Fernández Huerta translated the Flesch formula some

decades ago [17], replacing coefficient 84.6 by 60.0, in a very quoted paper that, nevertheless, provided no empirical proof. The change means that to obtain the same Flesch readability index, WPS must be equal in English and Spanish, but SPW must be 1.44 times higher in Spanish. Another similar index is Flesch-Kinkaid [29], which follows the formula $R_{Kinkaid} = 0.39 \times WPS + 11.8 \times SPW - 15.59$. This index yields a result between 0 and 12, corresponding to the school level a text is recommended for in the American education system; thus, contrary to the former, the corresponding step function is decreasing: the lower the Flesch-Kinkaid index, the more readable the text.

Readability indexes have been applied to requirements since the first works on quality metrics appeared [52]. However, the use of these indexes has received also numerous criticisms [28, 40, 41]. Indeed, we may expect that requirements consist of short and simple sentences; but, being technical texts addressed to professionals, it is completely acceptable that they contain numerous long words. Therefore, it is dubious that readability indexes conceived to qualify school readings may be applicable to the requirements of an information system. In any case, the more closely related desirable property with these indicators is understandability.

4.1.3. Punctuation. Another interesting measure, also related to requirements understandability, and relatively easy to obtain, is the number of *punctuation signs* per sentence, divided by sentence length: adequate punctuation is essential to sentence understandability, and punctuation excess makes the text more difficult to read, maybe pointing out that the sentence must be split in two or more sentences. Lack of punctuation in long sentences is equally pernicious, therefore the step function is convex.

4.1.4. Acronyms and abbreviations. The excess of *acronyms* (NASA, E.S.A., etc.) and *abbreviations* ('no.' for 'number') can be used as an indicator of lack of quality. Acronyms are easy to detect (words formed entirely or mostly with capital letters), even though maybe NOT in a fully deterministic way (for example, if the author wanted to emphasize a certain word by means of the use of capital letters, as in this very sentence). The use of acronyms is not pernicious if they are well defined in a glossary; however, the excess of acronyms may make a requirement less understandable. Abbreviations can be detected as words ending with a period ('.') that are not at the end of a sentence; equally, detection is not deterministic (if the abbreviation is at the end of the sentence, or if the next sentence wrongly starts with a lowercase letter). The use of abbreviations is even less acceptable than the use of acronyms, and it is usually not justified.

4.2. Lexical indicators

The morphological indicators we have studied in the previous subsection do not require any additional user-provided information to be computed. On the contrary, the set of *lexical* indicators we group together in this subsection are characterized by measuring properties that use reference information, i.e. they compare the text of the requirements with several *user-defined lists of terms*. Except for this characteristic, they are indicators relatively easy to define and obtain.

4.2.1. Connective terms. In the first place, we can measure the frequency of several kinds of *connective terms and particles* that, generally, are necessary in any linguistic construct, but whose abuse may imply a diminution of quality, especially regarding the properties of atomicity, precision, abstraction, understandability and unambiguity. In all of them the step transformation function should be decreasing, since the intention is to avoid an abuse, being the moderate use right.

- Number of *copulative-disjunctive terms*: they are potentially related with the lack of atomicity in requirements. The use of some conjunctions, particularly and/or, may denote that, instead of a single requirement, they are in fact two different requirements (example: the user will be authenticated *and* will visualize the state of his/her account). However, the use of copulative or disjunctive terms may be perfectly legitimate when the intention is to precisely specify a logical condition [27, 28], so that this measure must be defined and handled with care (example: the boiler will stop if it has been working for 10 minutes or if the water temperature is higher than 60 ºC). On the other hand, the different uses of conjunction or (mainly disjunctive-inclusive, disjunctive-exclusive, and explicative) may cause a lack of precision, and thus ambiguity or bad understandability.
- Number of *negative terms*: accumulating particles such as not, no, neither, never, nothing, nowhere, etc., may make the sentence more difficult to understand, besides increasing the risk of logical inconsistencies; it affects specially the desirable property of understandability.
- Number of *control flow terms*: in a similar way as the previous ones, they are conjunctions such as while, when, if... then, whose abuse probably points to an excess of detail in the way of specifying the flow of control of a process or function, going beyond the limits of a must-

be abstract requirements specification. This indicator is specially related to the desirable property of abstraction, and maybe with atomicity.

- Number of *anaphorical terms*: i.e. terms that are in the place of other terms; they are typically personal pronouns (it), relative pronouns (that, which, where), demonstrative pronouns (this, those), etc. Even with a grammatically impeccable usage, anaphors increase the risk of imprecisions and ambiguities in texts of technical character.

4.2.2. Imprecise terms. In the second place, some of the typical defects that should be eliminated [1, 52] are related to the use of imprecise terms that introduce ambiguities in the requirement. Therefore, the comparison with lists of forbidden terms can provide also interesting metrics. The use of imprecise or subjective terms is always reproachable, thus in this case we are talking of decreasing step functions. We can group together the terms by the kind of imprecision that characterizes them:

- Quality: good, adequate, efficient, etc.
- Quantity: enough, sufficient, approximately, etc.
- Frequency: nearly always, generally, typically, etc.
- Enumeration: several, to be defined, not limited to, etc.
- Probability: possibly, probably, optionally.
- Usability: adaptable, extensible, easy, familiar, safe, etc.

4.2.3. Design terms. Finally, the abuse of terms closely related to *design or technology* (method, parameter, data base, applet) denotes a lack of abstraction in requirements. The corresponding step function is decreasing.

4.3. Analytical indicators

We call *analytical* a set of indicators that require a textual analysis of requirements by means of relatively complex linguistic tools and that, in general, do not yield fully deterministic results.

4.3.1. Verbal tense and mood. The usage of *verbal tense and mood*, which requires an analysis of verbal flexion, is highly dependent on the language the requirements are written in. Since the beginning of the related researches [52] it has been emphasized the utility of analyzing the usage of most typical verbal forms to express the action that constitutes the core of a requirement: present or future tense (the user visualizes the state of his/her account, the user will visualize the state of his/her account), and infinitive forms preceded by auxiliary verbs of obligation or possibility (in present or future tense: the user must visualize, the user has to visualize, the user will be able to visualize, the application allows visualizing). The *number of verbal forms* meaning obligation or possibility, often called *imperative forms* in the literature (with terminological imprecision we nonetheless accept), is a good indicator of requirements atomicity: an excess of imperative forms would indicate that the requirement is not atomic enough. Some research works [52] even seek to identify and count requirements from the analysis of imperative forms, when it happens that requirements are not explicitly individualized. On the other side, the *usage of conditional mood* instead of future tense is a typical mistake, which probably originates in the unconscious desire to express a lower level of need in the requirement. A good specification must express the core of the requirement in a pure form, and express separately its need in the form of an attribute (typically at three levels: essential, convenient, optional). It is also necessary to consider here the number of usages of *passive voice* (it is visualized): these usages tend to leave implicit the verbal subject, leading to a certain degree of imprecision.

The analysis of verbal forms has different peculiarities in each language. In the case of English, which most part of literature refers to, this analysis can be rather easy, since looking for a few key words provides a lot of information (shall, must, have/has to, can, should, could, etc.). In the case of Spanish and other similar Romanic languages, where verbal conjugation is a lot richer and has a multitude of exceptions to basic grammar rules, the analysis is much more costly. However, the obtained benefit supports their use for requirements quality metrics.

4.3.2. Domain terms. Another relevant quality measure is the *number of domain terms* (nouns as well as verbs) in each requirement. Obtaining this measure requires the normalization of different grammatical versions of the terms (verbal conjugation, singular/plural for nouns, and even gender for nouns in languages such as Spanish) to find the canonical form defined in the domain; tools are also required to detect meaningful compound terms. Once nouns and verbs have been normalized, they are compared with the terms defined in the domain, which can be organized as a simple glossary of terms, or in more structured and complex forms such as thesauri or ontologies. The number of domain terms in each requirement should be neither too high (which would indicate a loss of atomicity) nor to low (which would indicate imprecise requirements

because they do not employ domain terms, or else an ill-constructed domain, which does not sufficiently cover the terms employed in the requirements).

4.4. Relational indicators

In contrast to the indicators seen so far, the indicators we denominate *relational* do not measure properties of individual requirements, but rather structural properties of the requirements set. The relationships among requirements, in general, cannot be automatically determined, so that for the purpose of quality measures we will assume these relationships have been already defined in the representation of requirements. We can distinguish, among others, the following relational measures:

4.4.1. Number of *versions* of a requirement. An excessive number of versions, especially if many of them come from recent dates, is a good indication of the volatility or instability of the requirement, which can be due to multiple causes (bad understanding of client needs, insecurity of the client about what he or she really wants, etc.). The quality of requirements demands, among others, a high degree of stability, which directly influences both validability and verifiability.

4.4.2. Degree of *nesting*. If requirements are hierarchically structured (either because some requirements are subordinated to other, or because they are grouped together in packages and subpackages of requirements), we can obtain measures such as the depth level of the hierarchy, and particularly the degree of nesting, i.e. the average number of elements subordinated to a given one. A traditional criterion to ease the understandability in information organization is that the degree of nesting should be neither too low nor too high.

4.4.3. Number of *dependencies* of a requirement. That is, dependencies towards other requirements or, in general, towards other artefacts in the development process (we understand that B depends on A when B requires the presence of A, or when a change in A may affect B). In the same way as we try to minimize dependencies among artefacts in design to ease maintenance and reuse, we must also try to minimize the number of dependencies among requirements. The excessive number of dependencies possibly denotes a lack of atomicity, understandability and traceability, thus affecting the three final desirable properties (validability, verifiability and modifiability). Nevertheless, the existence of dependencies is natural, and the fact that they are not reflected in the representation of requirements would indicate an insufficient analysis, rather than their non-existence. Therefore, we can expect a moderate number of dependencies per requirement, neither too high nor too low (convex step transformation function).

4.4.4. Number of *overlappings* among requirements. Understood as the number of requirements that talk about the same subject as a given one. Here we can distinguish between *conflict* when there is a contradiction between two requirements, *redundancy* when there is a needless repetition (which implies as well the risk of contradiction), and simple *coupling* when it is none of the former cases (and which implies some kind of dependency relationship). Linguistic processing tools (for example, by terminological co-occurrence, especially when a domain of reference exists) can help to detect overlappings among requirements, even though the engineer must determine the concrete kind of overlapping. Obviously, conflicts and redundancies must be eliminated; in contrast, it is natural that there are simple couplings, though the amount should not be too high, because that would indicate a lack of atomicity. Overlappings in general affect understandability, unambiguity and traceability of requirements.

Table 1 summarizes the measurable indicators we have presented so far, together with the desirable properties directly or indirectly related to each one of them.

Indicators	Function	Desirable properties										
		Atomicity	Precision	Completeness	Consistency	Understandability	Unambiguity	Traceability	Abstraction	Validability	Verifiability	Modifiability
Size	Convex	X	●	●	●	●	●	●	●	●	●	●
Readability	Incr./Decr.					X	●			●	●	●
Punctuation	Convex					X	●			●	●	●
Acron. & Abbrev.	Decreasing					X	●			●	●	●
Connective terms	Decreasing	X	X	●	●	X	X	●	X	●	●	●
Imprecise terms	Decreasing		X	●	●	●	X			●	●	●
Design terms	Decreasing								X		●	●
Imperative verbs	Convex	X	●	●	●	●	●	●	●	●	●	●
Conditional verbs	Decreasing		X	●	●	●	●			●	●	●
Passive voice	Decreasing		X	●	●	●	●			●	●	●
Domain terms	Convex	X	X	●	●	●	●	●	●	●	●	●
Versions	Decreasing									X	X	
Nesting	Convex					X				●	●	
Dependencies	Convex	X	●	●	●	X	●	X	●	●	●	●
Overlappings	Decreasing	X	●	●	●	X	X	X	●	●	●	●

Table 1 Measurable indicators and related desirable properties (X = direct influence, ● = indirect influence)

5. Quality indexes per requirement and global quality measures

Once we have defined the quality indicators we want to apply and their corresponding step transformation functions, we can process the requirements by means of an automatic quality management tool. An adequate tool should in the first place provide concrete indications to improve the quality of requirements, distinguishing between commands to correct bad results and recommendations to improve medium ones. For example, you *must* reduce the number of imprecise terms, if the result has been Bad; you *should* increase the number of domain terms, if the result has been Medium; etc. In the second place, the tool should provide a *quality index* for each individual requirement, which requires aggregating in some way the results obtained for each indicator. With this index we can classify requirements and thus concentrate on solving the problems of the worst ones. Finally, it would be also useful to provide a *global quality measure* over a certain requirements set, which requires aggregating the quality indexes obtained for individual requirements. This global measure could serve to evaluate the quality of a project, to support the quality improvement of the requirements produced by a certain engineer, etc.

5.1. Quality index of a requirement

It is reasonable to think that some indicators are more important than others to evaluate the quality index of a requirement, so that in order to compute the *weighted average* we have to define the *relative weight of each indicator*. Besides, we have to assign numerical values to the nominal values (Bad, Medium, Good) provided by the step transformation functions of each indicator: for example, 0-1-2. Another possible approach consists in assuming that the aggregated measure is not the weighted average of the indicators, but the *minimum value* among all of them: a requirement is Bad if one indicator has that value, even though the others have a higher value. We can even adopt a *mixed approach*: we define some indicators as *prevailing*, which means that, if one of these indicators is Bad, then the aggregated result is Bad independently of the others (note this is not the same as taking the minimum of prevailing indicators, since the minimum could be Medium); if no indicator is prevailing and Bad at the same time, then we take the weighted average as the aggregated measure.

On the other side, it is not enough to assign numerical values to nominal values; we need also to interpret the quality index as an aggregated result. If nominal values are translated into 0-1-2, as we are assuming by default, then the aggregated value (average, minimum or mixed) will be contained within the interval [0, 2]. We then define three sub-intervals within this interval, which

are not necessarily equal in size. For example, suppose the aggregated quality index per requirement is Good if there are at least 50% of requirements with individual result Good, at least 50% with result Medium, and none of them is Bad (or another equivalent distribution on average); and the quality index is Medium if no indicator is Good, at least 50% of indicators are Medium, and the remaining 50% is Bad; then the sub-intervals we have to define to interpret the aggregated quality index per requirement are: Bad in [0, 0.5), Medium in [0.5, 1.5) and Good in [1.5, 2]. Obviously, we could establish a greater number of sub-intervals if it is judged necessary.

Summing up, a quality measurement tool based on indicators could define (see Table 2):

- Whether the *aggregation function* to compute the quality index per requirement is the weighted average of indicators, the minimum of indicators, or the weighted average taking into account prevailing indicators.
- The *numerical values of nominal values* of indicators (Good, Medium, Bad), needed to compute the aggregated value of each requirement.
- The *sub-intervals to interpret aggregated values* of the quality index.

Also, for each used indicator we could define:

- The kind of step transformation function (increasing, decreasing, convex, concave).
- The intervals of the step function (x_1-x_2 or x_1-x_2-x_3-x_4, according to the kind of function).
- The relative weight of the indicator to compute the aggregated value.
- Whether the indicator is prevailing over the others.

Finally, it can be useful to disable an indicator without need of losing its definition.

With the parameterization of intervals given as an example in Table 2, the indicator based on the number of imperative forms, having a convex step function, is considered Good (=2) if there are 1, 2 or 3 imperatives per requirement, Medium (=1) if there are 4, and Bad (=0) if there are 0, 5 or more (the second interval, which would yield also a Medium value, has been canceled because it has been defined as 1-1). The other indicators are defined analogously. The quality index per requirement is calculated according to the aggregation function selected, in this case the weighted average (therefore, the fact that two indicators have been defined as prevailing has no effect). Note that defining an indicator as prevailing does not necessarily imply that its weight should be higher: this gives the user more freedom to find the best tool configuration. Supposing the four enabled indicators gave the results Good-Medium-Bad-Good for a given requirement, then the average would be 1.17, falling within the interval [0.5, 1.5), therefore the requirement would be classified as Medium by its quality index.

Aggregation function	☒ Average ❑ Minimum ❑ Average with prevailing indicators						
Nominal values	Bad = 0		Medium = 1		Good = 2		
Aggregated values	Bad = [0, 0.5)		Medium = [0.5, 1.5)		Good = [1.5, 2]		
Indicator	**Category**	**Kind**	**Intervals**	**Weight**	**%**	**Prevailing**	**Enabled**
Size	Morphological	Convex	10-40-80-200	1	.17	No	Yes
Readability	Morphological	Increasing	20-50	1	.17	Yes	Yes
Imprecise	Lexical	Decreasing	0-2	2	.33	Yes	Yes
Imperative	Analytical	Convex	1-1-4-5	2	.33	No	Yes
Overlappings	Relational	Decreasing	3-6	1	.00	No	No

Table 2 An example of quality measurement parameterization by means of indicators

The tool user would have the task to define all these parameters, according to his or her own yardstick, or the one imposed by the organization's quality department. As it can be easily observed, the greatest difficulty consists in determining the step function intervals, as well as the relative weights of indicators, since we can easily finish by assigning arbitrary values, with no other fundament than the user's intuition. The *tuning* of parameters in a tool such as the one explained above, however, is out of the scope of this work.

5.2. Global quality measure of a set of requirements

As we have mentioned above, it is also useful to provide a *global quality measure* over a certain requirements set. This global measure serves to evaluate the quality of a whole project, of the requirements produced by a certain engineer, etc. It can be used to find a certain set of requirements in need of special attention and to concentrate on solving local problems. Nevertheless, let's remind the reader that the main goal of quality measurements is not to obtain a

global numerical measure (*the policeman's view*), but to point out defects and help to remove them (*the counselor's view*) [26].

The most obvious way to compute the global measure is obtaining the quality index per requirement and then averaging for all requirements. The measurement provides a numerical result in the range [Bad, Good], i.e. [0, 2] with the numerical values we are using by example. The result can be interpreted as Good, Medium, Bad, employing the same intervals defined to interpret the quality index per requirement.

5.3. Summary of the procedure to obtain quality measurements

The full procedure consists of the following steps:

1. Preparatory steps:
   1.1. Define the different lists of forbidden terms used to compute *lexical indicators*, and the list of *domain terms* (nouns and verbs).
   1.2. For each indicator, define its *step transformation function* and associated parameters.
   1.3. Choose an *aggregation function* to compute the quality index per requirement.
   1.4. Select a *set of requirements* to be evaluated.
2. Computational steps:
   2.1. Obtain the *primitive numerical measures* by counting textual features of the requirements, such as number of words, number of ambiguous terms, etc.
   2.2. Transform these primitive values into *quality indicators* by means of the corresponding step functions; use the results to provide concrete recommendations to improve each requirement.
   2.3. Compute the *quality index per requirement* using the aggregation function; use this value to classify/sort the requirements according to their quality index.
   2.4. Compute the *global quality measure* of the requirements set; use this value to compare different projects, assess global quality improvement, etc.

Step 2.1 is by far the most time-consuming one, so that the capacity to perform it off-line is most convenient. The recommendations provided in step 2.2 constitute the main added value that can be obtained from a tool like this: this must be emphasized over the possibility to obtain quality indexes per requirement and, especially, global quality measures.

6. Tool support and evaluation of the theoretical framework

As an important piece of the innovation process, once the theoretical framework is mature enough, it is possible (and necessary) to build a tool that can prove the selected approach is valid through industrial validation. *RQA Requirements Quality Analyzer* [48] is a tool developed by The Reuse Company in close collaboration with the Knowledge Reuse Group at Universidad Carlos III de Madrid [31]. This tool fully implements the quality measurement principles and techniques that are described in the previous Sections of the paper. In the following subsections we present the tool, we compare it with other requirements quality tools, and we give information about its actual use by industry.

6.1. Description of the RQA tool

The Requirements Quality Analyzer (RQA) was not designed to be a general requirements management tool, but to operate in collaboration with other tools. Therefore, RQA receives the requirements as input data and computes quality metrics and recommendations as output. Based on this architectural design, the main features of this tool are:

- Developed for the Microsoft Windows platform.
- Input requirements accepted in Microsoft Excel format, both in English and in Spanish.
- Also fully connected to the DOORS repository of requirements.
- Quality metrics computed in line with the principles explained in this paper.
- Configuration of metrics according to the organization needs: lists of forbidden terms, list of domain terms, definition of transformation and aggregation functions, etc.
- Quality metrics can be computed in batch mode to save user's time.
- Besides metrics for individual requirements, the tool computes block metrics for a set of requirements: number of overlappings among requirements with semantic proximity, and number of requirements using incompatible measurement units (such as length in meters/feet).

The tool is intended to have the following main uses:

- Analysis of the quality of requirements of customers in a call to tender.
- Analysis of the quality of requirements before a business review, peer review or inspection, enabling the actor to concentrate on the substance rather than try to understand the form.
- Requirements authoring assistance to help systems engineers to write good requirements in a first shot.
- Assistance to establish traceability links or to find overlapping requirements in a single specification, by detecting similarity on requirements.

The tool has been conceived to be used by different collaborative roles in the organization:

- The *Quality Department* defines quality metrics, weights and thresholds according to the organization's approved best practices.
- The *Project Manager* checks the quality of his/her projects, and identifies the most problematic metrics that require additional training.
- The *Requirements Engineer* is assisted by the tool to write better requirements right the first time.

Establishing too stringent thresholds from the start is inadequate, since it would clash against the opposition of requirements engineers. It is better to learn from experience and proceed by gradual adjustments. Therefore, the tool promotes a continuous cycle of improvement following a PDCA process (plan-do-check-act [12]), where the desired quality features are smoothly approached, or even by quantum leaps [2]:

- *Plan*: Define the scope and expected output of the process. Take initial measurements. Identify initial metrics, weights and thresholds. Involve tool users in the process.
- *Do*: Deploy the tool. Train tool users. Introduce the tool into the engineering process, first on a small scale if possible. Take actual measurements.
- *Check*: How is requirements quality evolving? Do we need additional training?
- *Act*: Perform additional training if needed. Define new metrics and adjust current thresholds.

Figures 4 to 7 show some representative snapshots of the tool.

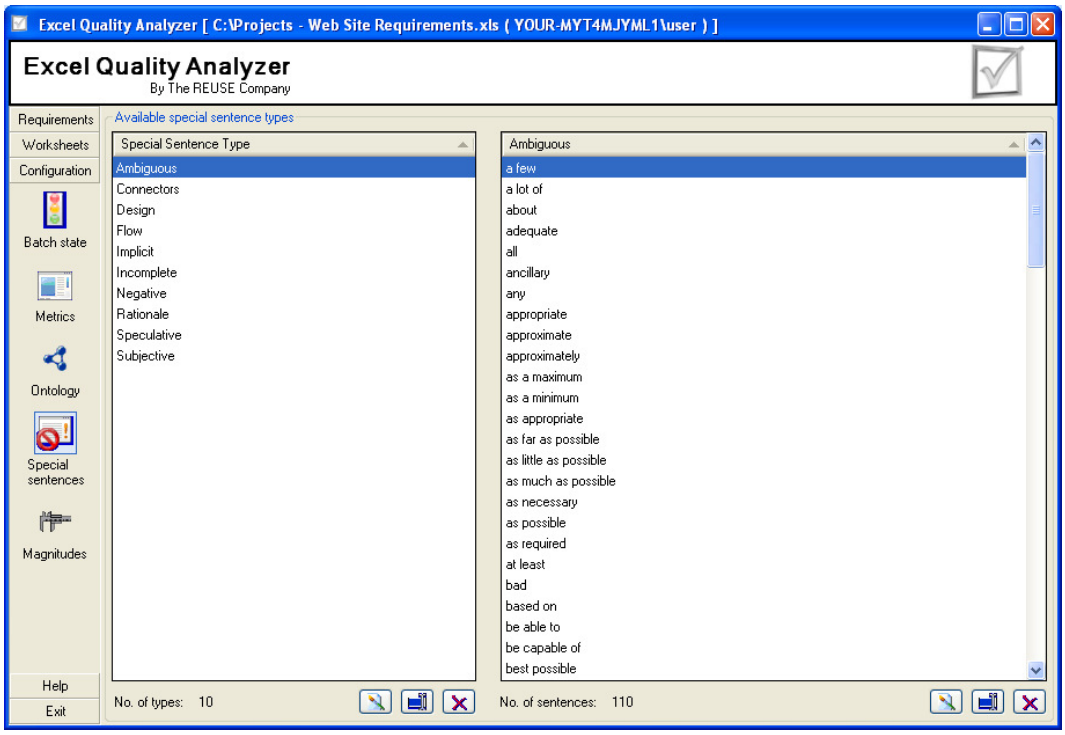

Fig. 4 Configuration of metrics: definition of the list of *ambiguous sentences*

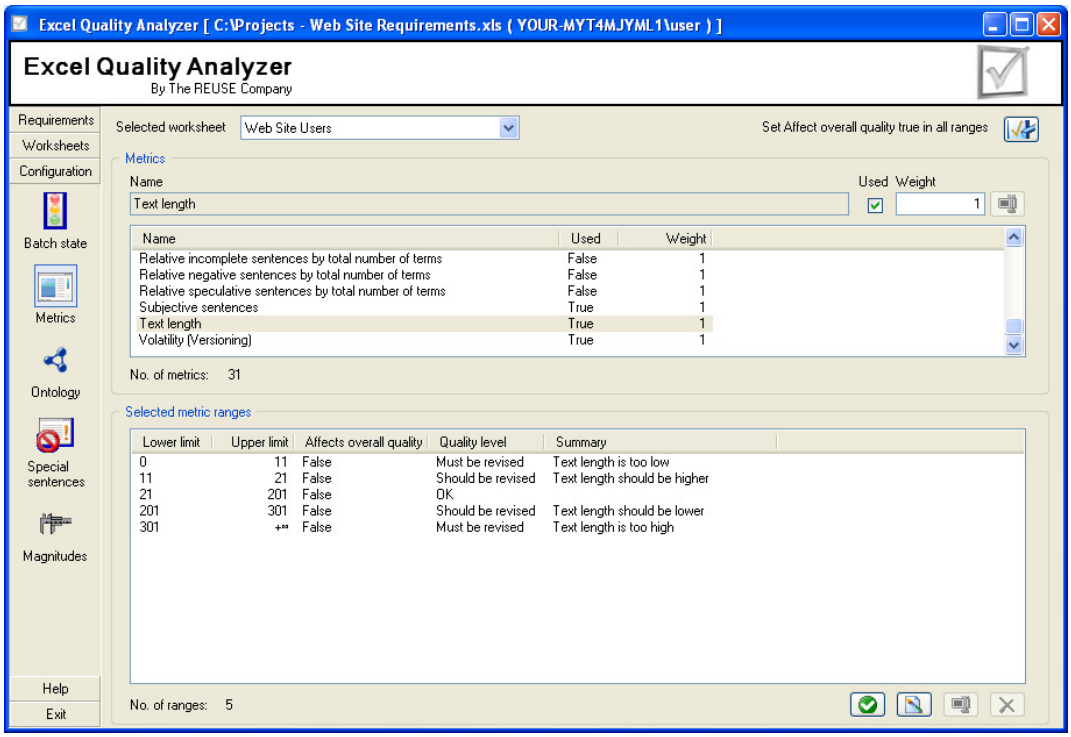

Fig. 5 Configuration of metrics: definition of the step transformation function for the *text length* indicator. Text length is here considered optimal between 21 and 200 words

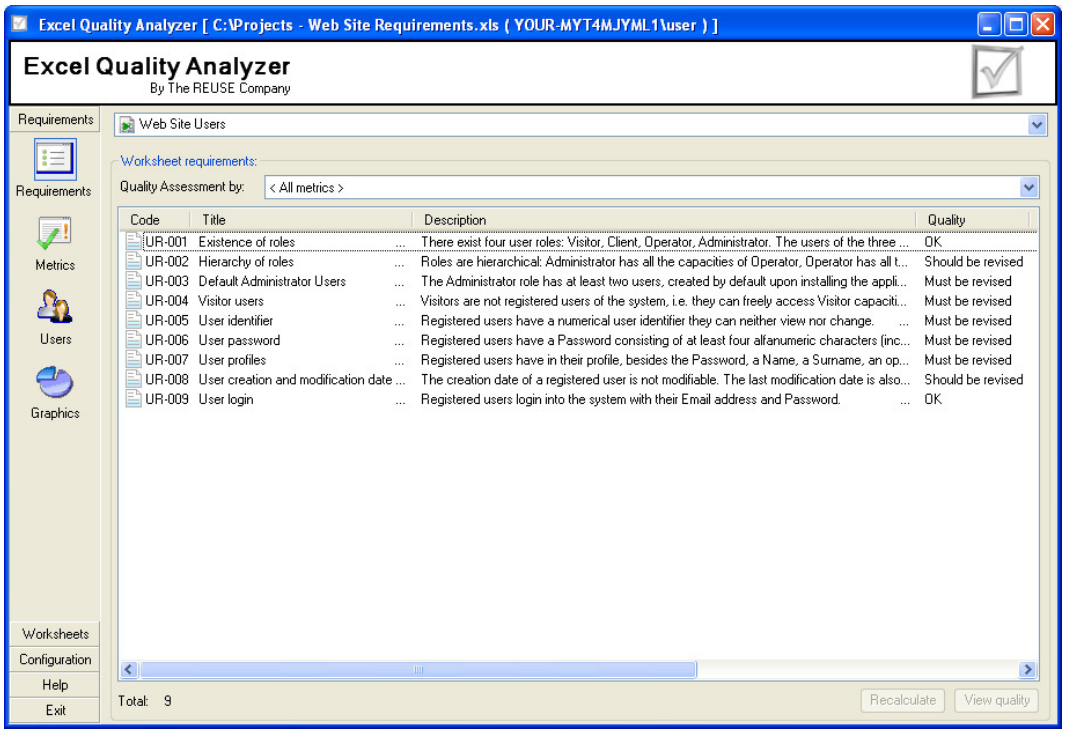

Fig. 6 Quality assessments obtained for a set of requirements: each requirement is marked as Good (OK), Medium (Should be revised) or Bad (Must be revised)

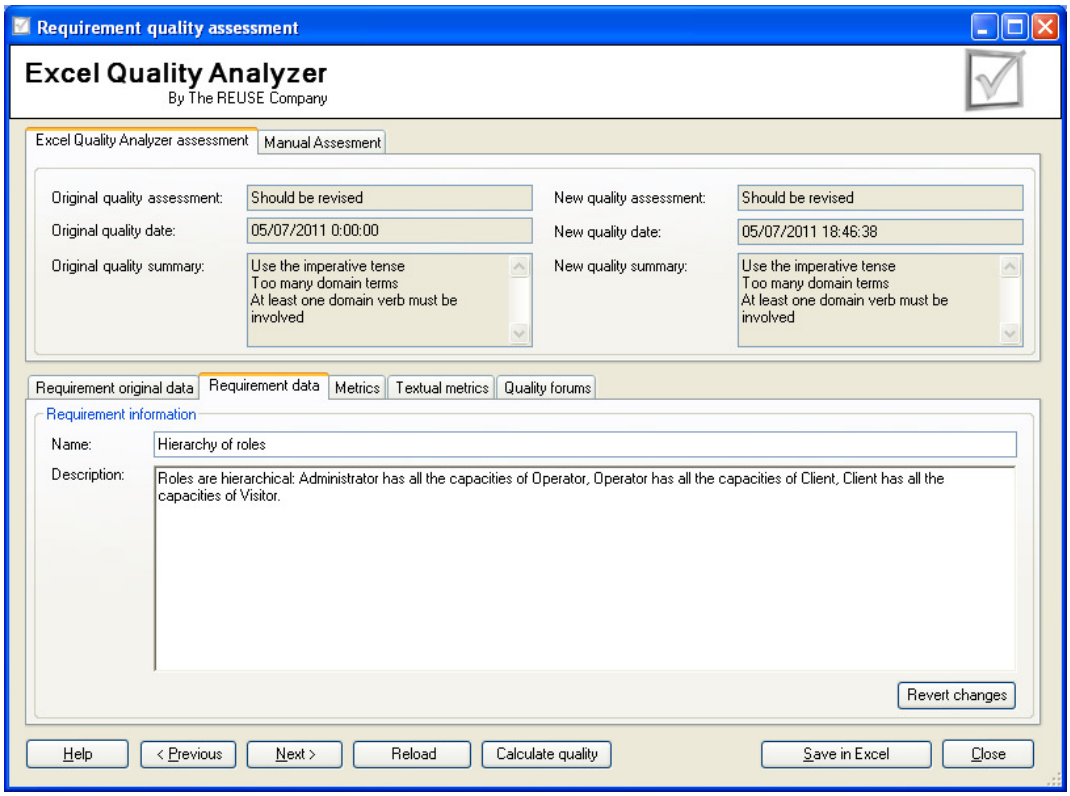

Fig. 7 Concrete recommendations to improve the quality of requirements: this particular requirement has been found of only Medium quality because no imperative verbal forms are used, too many domain terms are used, and no domain verb is used

6.2. Comparison with other requirements quality tools

This subsection summarizes the information we have found about tools that deal in one way or other with requirements quality. Our results are based on the external survey performed in June 2010 by the RAMP project (Requirements Analysis and Modeling Process). This project is a joint initiative between three major industrial companies belonging to different domains (EADS, RENAULT, EDF), two smaller companies providing service consulting and tools (ADN, Cortim) and four academic & research institutions (IRIT, ENSTA, INRIA, Paris 1 University/Sorbonne). The goal of the RAMP project is to improve the efficiency and quality of requirements expressed in natural language during the development of complex systems [14].

Having detected that most organizations do not use dedicated tools to support requirements quality verification, they performed an assessment over a set of tools and arrived at the conclusion that RQA (Requirements Quality Analyzer) by TRC (The Reuse Company) is the leading tool [15]. From an initial set of 15 tools, 8 were selected to perform an in-depth assessment (see Table 3). The results of the survey were:

- RQA was considered the most promising tool, followed by LEXIOR as its closest competitor.
- Requirements Assistant and ARM lack integration with DOORS (a well-known and widespread requirements management tool, also evaluated in this survey).
- DESIRe was considered interesting but out of scope (more a support for requirements authoring).
- The remaining tools (QuARS, TigerPro and DOORS) did not comply with industrial expectations.

Tool	Company	Country	Use	Kind	Result
RQA [48]	The Reuse Company	Spain	Industrial	Commercial	Most promising as assessed so far
LEXIOR [11]	Cortim	France	Industrial	Commercial	Most important challenger
Requirements Assistant [45]	Sunny Hills	Holland	Industrial	Commercial	Not integrated with DOORS
ARM [35]	NASA GSFC	USA	Industrial	OpenSource	
DESIRe [21]	HOOD	Germany	Industrial	Commercial	Authoring Support Application
QuARS [49]	University of Pisa	Italy	Academic	Commercial	Academic Tool Service Oriented Business Model
TigerPro [50]	University of South Australia	Australia	Academic	OpenSource	
DOORS [23]	IBM	USA	Industrial	Commercial	Too complex Compliance with COTS Policy Cost of Maintenance

Table 3 Survey by the RAMP project of industrial or academic tools that deal with requirements quality. Tools are sorted top-down according to their assessment

6.3. Use of RQA in Industry

The RQA tool presented in this paper left the University labs and The Reuse Company in 2010, towards its commercialization in Europe. At present date (June 2011) the most relevant customers that have acquired the tool are EADS IW, ASTRIUM, ADN, SAGE Group and ST Microelectronics. The tool is also under evaluation by AIRBUS MILITARY and THALES AVIONICS.

The reception of the tool by its intended users can be summarized as follows: the engineers feel an initial fear that is completely defeated after the training process, transforming this feeling into confidence when the tool starts suggesting improvements at the requirements level. This initial fear is due mainly to a perception of the tool as a policeman instead of as a counselor, and to a lack of understanding of the use of ontologies for semantic/lexical analysis of requirements in RQA.

The users also perceive the benefits of using RQA:

- The final quality of the written requirements increases, positively affecting the Verification and Validation process.
- Once the organization has defined them, the quality thresholds become a valuable asset and the kernel for process improvement through metrics.
- Engineers learn to write requirements in the way the organization wants while they are doing the actual job (*learning on the job* [33]).
- The use of global metrics in huge projects (tens of thousands of requirements) has shown to be very successful to find sets of requirements in need of special attention and to concentrate on solving local problems.

7. Conclusion

The introduction and use of quality metrics in a software development enterprise environment requires considering at least two aspects. In the first place, trying to obtain all these measures by hand would be illusory, therefore automated tools become indispensable. In the second place, a gradual implementation must be tried to avoid the refusal of requirements engineers, who should see an aid in quality management tools, not a policeman mechanism to prosecute or penalize. Graduality must be taken into account, too, when deciding which metrics and thresholds to apply, without trying to be too demanding from the beginning.

Albert Einstein is often quoted as having said: “Not everything that can be counted counts, and not everything that counts can be counted”. In fact the words must be credited to sociologist William Bruce Cameron [9]. Even though quantitative measurement constitutes one of the foundations of modern empirical science, numerical measures must be used with care and wisdom. Assessing the quality of requirements (or whatever engineering artefact) requires human judgment. This judgment can be assisted, but not replaced, by objective measurements. Nevertheless, we think an automated tool that provides low-level quality indicators can provide valuable hints to improve high-level quality features of requirements.

Acknowledgments

This research is supported through the Spanish Ministerio de Ciencia y Tecnología, Project TIN2007-67153, “SEMSE: SEmantic Metadata SEarch” (“Desarrollo de un sistema de

recuperación conceptual mediante niveles semánticos en la representación de esquemas de metadatos").

References

[1] I. Alexander, R. Stevens. *Writing Better Requirements*. Addison-Wesley, 2002.

[2] L.J. Arthur. "Quantum improvements in software system quality". *Communication of the ACM* 40(6):46-52, 1997.

[3] D.M. Berry, A. Bucchiarone, S. Gnesi, G. Lami, G. Trentanni. "A new quality model for natural language requirements specifications". *Proceedings of the 12th International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ-06)*. Luxembourg, June 5-6 2006. Held in conjunction with CAiSE'06.

[4] E. Braude. *Software Engineering, An Object Oriented Perspective*. John Wiley & Sons, 2001.

[5] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[6] A. Bertolino, A. Fantechi, S. Gnesi, G. Lami, A. Maccari. "Use Case Description of Requirements for Product Lines". *Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL'02*. Held in conjunction with RE'02. Essen, Germany, September 9 2002.

[7] F.P. Brooks. "No Silver Bullet. Essence and Accidents of Software Engineering". *IEEE Computer* 20(4):10-19, April 1987. Reprinted in: F.P. Brooks. *The Mytical Man-Month, Essays on Software Engineering.* Addison-Wesley, 1995 (20th Anniversary Edition).

[8] A. Bucchiarone, S. Gnesi, P. Pierini. "Quality Analysis of NL Requirements: An Industrial Case Study". 13th IEEE International Requirements Engineering Coference, pp. 390-394. Paris, France, August 29 – September 2, 2005.

[9] W.B. Cameron. *Informal Sociology: A Casual Introduction to Sociological Thinking*. Random House, 1963.

[10] M. Christel, K. Kang. *Issues in Requirements Elicitation*. Carnegie Mellon University, Software Engineering Institute. Technical Report CMU/SEI-92-TR-012, 1992.

[11] Cortim. *Lexior* (http://www.cortim.com/lexior/).

[12] W.E. Deming. *Out of the Crisis*. MIT Press, 1986.

[13] F. Fabbrini, M. Fusani, S. Gnesi, G. Lami. "The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the Use of an Automatic Tool". *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, pp. 97-105, 2001.

[14] G. Fanmuy, J.-C. Roussel, R. Szczepaniak, C. Salinesi, A. Dauron, L. Picci, O. Hammami. "Requirements Analysis and Modeling Process (RAMP) for the Development of Complex Systems". *7th European Systems Engineering Conference (EuSEC 2010)*. Stockholm, Sweden, May 23-26, 2010.

[15] G. Fanmuy. "Requirements Verification in Industry". *17th International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ-2011), Industry Track*. Essen, Germany, March 28-30, 2011.

[16] A. Fantechi, S. Gnesi, G. Lami, A. Maccari. "Applications of linguistic techniques for use case analysis". *Requirements Engineering* 8(3):161-170, 2003.

[17] J. Fernández Huerta. "Medidas sencillas de lecturabilidad". *Consigna*, 214:29-32, 1959.

[18] R. Flesch. "A New Readability Yardstick", *Journal of Applied Psychology* 32:221-233, 1948. Reprinted in: Rudolf Flesch. *The Art of Readable Writing*. New York, 1949 (25th Anniversary Edition, Revised and Enlarged, Harper & Row, 1974).

[19] U. Frank. *Towards a Pluralistic Conception of Research Methods in Information Systems Research*. ICB-Research Report No. 7, 2006. Institute for Computer Science and Business Information Systems, University Duisburg-Essen. Retrieved May 19, 2011 from http://www.icb.uni-due.de/fileadmin/ICB/research/research_reports/ICBReport07.pdf.

[20] A.R. Hevner, S.T. March, J. Park, S. Ram. "Design science in information systems research". *MIS Quarterly*, 28(1), 75–105, 2004.

[21] HOOD. *DESIRe* (http://www.hood-group.com/en/products/tools/requirements-engineering/desirer/).

[22] M. Ibáñez, H. Rempp. *European User Survey Analysis*. European Software Institute, ESPITI Project, Technical Report TR95104, 1996.

[23] IBM. *Rational DOORS* (http://www-01.ibm.com/software/awdtools/doors/).

[24] IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications* (http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf).

[25] IEEE Std 982.2-1988. *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software* (http://ieeexplore.ieee.org/iel1/2596/983/00026479.pdf).

[26] L. James. "Providing Pragmatic Advice On How Good Your Requirements Are - The Precept 'Requirements Councillor' Utility", *Proceedings of the 9th INCOSE International Symposium*, Brighton, England, 1999.

[27] J.E. Kasser. "The First Requirements Elucidator Demonstration (FRED) Tool". *Systems Engineering* 7(3):243-256, 2004.

[28] J.E. Kasser, W. Scott, X.L. Tran, S. Nesterov. "A Proposed Research Programme for Determining a Metric for a Good Requirement". *The Conference on Systems Engineering Research*, Los Angeles, California, USA, 2006.

[29] J.P. Kincaid, R.P. Fishburne, R.L. Rogers, B.S. Chissom. "Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel". *Research Branch Report* 8-75, Naval Air Station, Memphis, Tennessee, USA, 1975.

[30] N. Kiyavitskaya, N. Zeni, L. Mich, D.M. Berry. "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications". *Requirements Engineering* 13(3):207-239, 2008.

[31] Knowledge Reuse Research Group, Universidad Carlos III de Madrid (http://www.kr.inf.uc3m.es).

[32] M. Marrero, S. Sánchez-Cuadrado, A. Fraga, J. Llorens. "Applying Ontologies and Intelligent Text Processing in Requirements Reuse", *First Workshop on Knowledge Reuse (KREUSE'08)*, pp. 25–29. Held in conjuction with 10th International Conference on Software Reuse (ICSR'08). Beijing, China, 2008.

[33] V.J. Marsick, M. Volpe (Eds.). *Informal learning on the job*. Berrett-Koehler Publishers, 1999.

[34] L. Mich, M. Franch, P.L.N. Inverardi. "Market research for requirements analysis using linguistic tools". *Requirements Engineering* 9(1):40-56, 2004.

[35] NASA Goddard Space Flight Center. *ARM Automated Requirement Measurement Tool* (http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php#tools).

[36] N.S. Latimer-Livingston. *Market Share: Requirements Management, Worldwide, 2003 (Executive Summary)*. Gartner Research, 1 July 2004 (http://www.gartner.com/DisplayDocument?ref=g_search&id=452522).

[37] P. Loucopoulos, V. Karakostas. *Systems Requirements Engineering*. McGraw-Hill, 1995.

[38] D. Popescu, S. Rugaber, N. Medvidovic, D.M. Berry. "Reducing Ambiguities in Requirements Specifications via Automatically Created Object-Oriented Models". *Proceedings of the 14th Monterey Workshop on Requirements Analysis*. Monterey, CA, USA, September 10-13 2007. Springer LNCS 5320, pp. 103-124.

[39] R.S. Pressman. *Software Engineering A Practitioner's Approach*. McGraw-Hill Pub Co., Sixth Edition 2005.

[40] J.C. Redish, J. Seizer. "The Place of Readability Formulas in Technical Communication". *Technical Communication* 32(4):46-52, 1985.

[41] J.C. Redish. "Readability Formulas Have Even More Limitations Than Klare Discusses". *ACM Journal of Computer Documentation* 24(3):132-137, 2000.

[42] L.H. Rosenberg. "Generating High Quality Requirements". *Proceedings of the AIAA Space 2001 Conference and Exposition*, AIAA Paper 2001-4524. American Institute of Aeronautics and Astronautics, Albuquerque, NM, August 28-30, 2001.

[43] E. Sadraei, A. Aurum, G. Beydoun, B. Paech. "A field study of the requirements engineering practice in Australian software industry". *Requirements Engineering* 12(3):145-162, 2007.

[44] I. Sommerville. *Software Engineering*, 7th ed. Pearson-Addison Wesley, 2004.

[45] Sunny Hills. *Requirements Assistant* (http://www.requirementsassistant.nl/).

[46] The Standish Group, *Chaos Report*, 2003 (http://www.standishgroup.com/).

[47] *SWEBOK Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2004 (http://www.computer.org/portal/web/swebok).

[48] The Reuse Company. *RQA Requirements Quality Analyzer* (http://www.reusecompany.com/index.php?option=com_content&view=category&layout=blog&id=171&Itemid=75&lang=en).

[49] University of Pisa. *QuARS Quality Analyzer for Requirement Specifications* (http://quars.isti.cnr.it/).

[50] University of South Australia. *TigerPro* (http://www.therightrequirement.com/TigerPro/TigerPro.html).

[51] V.K. Vaishnavi, W. Kuechler. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach Publications, 2008.

[52] W.M. Wilson, L.H. Rosenberg, L.E. Hyatt. "Automated Analysis of Requirement Specifications". *Proceedings of the 19th International Conference on Software Engineering-ICSE'97*, May 17-23, 1997, Boston, Massachusetts, USA: 161-171.

[53] R. Winter. "Design science research in Europe". *European Journal of Information Systems* 17:470–475, 2008.

[54] R. Zeist, P. Hendriks. "Specifying software quality with the extended ISO model". *Software Quality Journal* 5(4):273-284, 1996.