Moreno, V., Génova, G., Parra, E. et al. Application of machine learning techniques to the flexible assessment and improvement of requirements quality. *Software Qual J* 28, 1645–1674 (2020).

DOI: 10.1007/s11219-020-09511-4

Application of machine learning techniques to the flexible assessment and improvement of requirements quality

Valentín Moreno, Gonzalo Génova (ORCID 0000-0003-0299-286X)*, Eugenio Parra, Anabel Fraga
vmpelayo@inf.uc3m.es, ggenova@inf.uc3m.es, eparra@kr.inf.uc3m.es, afraga@inf.uc3m.es
Knowledge Reuse Group, Departamento de Informática, Universidad Carlos III de Madrid
Avda. Universidad 30, 28911 Leganés (Madrid), Spain
* Corresponding author, ggenova@inf.uc3m.es, +34 91 624 8846

Keywords: requirements quality; machine learning; automatic classification; automatic improvement; experts' judgment; flexible assessment.

Abstract

Context: It is already common to compute quantitative metrics of requirements to assess their quality. However, the risk is to build assessment methods and tools that are both arbitrary and rigid in the parameterization and combination of metrics. Specifically, we show that a linear combination of metrics is insufficient to adequately compute a global measure of quality.

Objective: In this work we propose to develop a flexible method to assess and improve the quality of requirements that can be adapted to different contexts, projects, organizations and quality standards, with a high degree of automation.

Method: The domain experts contribute with an initial set of requirements that they have classified according to their quality, and we extract their quality metrics. We then use machine learning techniques to emulate the implicit expert's quality function. We provide also a procedure to suggest improvements in bad requirements.

Results: We compare the obtained rule-based classifiers with different machine learning algorithms, obtaining measurements of effectiveness around 85%. We show as well the appearance of the generated rules and how to interpret them.

Conclusion. The method is tailorable to different contexts, different styles to write requirements, and different demands in quality. The whole process of inferring and applying the quality rules adapted to each organization is highly automated.

Acknowledgments

This research has received funding from the CRYSTAL project – Critical System Engineering Acceleration (European Union's Seventh Framework Program FP7/2007-2013, ARTEMIS Joint Undertaking grant agreement n° 332830); and from the AMASS project – Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems (H2020-ECSEL grant agreement nº 692474; Spain's MINECO ref. PCIN-2015-262). This research would not have been possible without the support and help of Gauthier Fanmuy, expert in Requirements Engineering and Model-

Based Systems Engineering, member of INCOSE (International Council on Systems Engineering) and AFIS (Association Française d'Ingénierie Système).

1. Introduction: requirements quality

Requirements Engineering is a "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" [Loucopoulos & Karakostas 1985]. Requirements engineers are the professional specialists who elicit and write the requirements, and as such they are often called also 'authors of requirements' or even 'requirements authors' [INCOSE 2012, Terzakis & Gregory 2016, Gregory & Terzakis 2017]. Organizations developing software for critical sectors like aerospace, automotive, and medical systems need to apply process requirements coming from different sources: industrial standards, customer-provided requirements, and procedures from internal quality management systems [Eito-Brun & Amescua 2017]. As it has been often pointed out [Brooks 1987, The Standish Group 2015, IEEE Computer Society 2014], most of the defects in the delivered software originate in a deficient requirements analysis, and they are generally the most difficult and costly to repair [Fanmuy et al. 2011]. That is why 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*. 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.

Since the very beginning of quality measurement in requirements engineering [Wilson et al. 1997], researchers have found that the need of communication among all stakeholders requires that the privileged form to express requirements is natural language, as opposed to formal languages that are more or less inaccessible, mainly to clients. Therefore, the use of linguistic techniques and tools may perform a crucial role in providing support for requirements analysis [Mich et al. 2004] and, in particular, in order to obtain quality metrics. Besides, the large quantity of requirements in many projects, as well as the different roles involved in their specification (users or clients, analysts, designers, developers, testers, etc.), recommend the use of guides [Hooks 1993, Magee & Tripp 1997, Rosenberg & Linda 2001, Alexander & Stevens 2002, Turk 2006, Bøegh 2008] and standards [ESA 1995, IEEE 1998, ISO/IEC 2007, INCOSE 2012] to achieve high quality from the start.

In order to obtain quality metrics, we must first define what we understand as good or bad quality of a requirement or set of requirements. In a previous work we have distinguished between *qualitative* desirable properties of requirements, dependent on subjective judgment, and *quantitative* measurable indicators, based on objective characteristics of requirements [Génova et al. 2013]. We synthesized the different lists of *desirable properties* that can be found in the literature in three hierarchical levels:

- First level: validability, verifiability and modifiability.
- Second level: completeness, consistency, understandability, unambiguity, traceability and abstraction; or, more synthetically, CCC (completeness, consistency, correctness).
- Third level: precision and atomicity.

These properties depend on subjective judgment, which does not mean that they are arbitrary, but that they are not easy to quantify. Therefore, we need to define a series of *measurable 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: the size indicator affects the desirable properties of the requirement, particularly atomicity, and all the others through atomicity. Equally, we can measure the number of imperative verbal forms, the number of domain terms, the number of ambiguous expressions, and so on. In this work we consider only metrics related to correctness of individual requirements, leaving global properties of the set (i.e. consistency and completeness) for future research.

Summing up, we can compute a set of quantitative metrics of textual requirements, and through them we can assess the quality of requirements. However, the risk of this approach is to build assessment methods and tools that are both arbitrary in the parameterization of metrics and rigid in the combination of metrics to evaluate the different properties. This is why we propose in this work to develop *a flexible assessment method* that can be adapted to different contexts, with a high degree of automation. The method consists basically in the emulation of the experts' judgment on quality through artificial intelligence techniques: first, obtain the expert's implicit quality function through machine learning; and, second, apply this function to automatically assess the quality of textual requirements.

Our approach to emulate the experts' judgment, as explained later in detail, is based on well-known machine learning techniques: we have a computer tool learn from a previous human-made classification of requirements according to their quality. Therefore, our work's intent is not to improve machine learning techniques, but rather to devise a novel application to the field of requirements quality assessment.

The rest of the paper is structured as follows. Section 2 reviews the related work on automatic measurement and improvement of requirements quality. Section 3 describes the research process we have followed for designing our solution approach. Section 4 justifies the need for the flexible assessment of requirements quality in different contexts, where the experts' judgment on quality is the best starting point to satisfy the needs and peculiarities of each project and organization. Section 5 shows that the regions of good and bad quality in the hyperspace of requirements metrics is not adequately modeled through a linear combination of metrics, but rather demand a more refined computation that could be solved with a versatile combination of rectangular hyper regions. Section 6 explains those well-known machine learning techniques that constitute the fundamentals of our method to build an automatic rule-based classifier that solves the problem of computing the concrete intervals for each metric and the combination of intervals of different metrics. Section 7 describes the initial data set from which a concrete set of rules has been generated, as well as the experiments performed to validate the rules, together with their results in effectiveness and efficiency. Section 8 analyzes the appearance of the obtained rule-based classifiers, and how to interpret the rules. Section 9 complements the flexible assessment of quality with a procedure to suggest improvements in the requirements that are classified with bad quality, providing a mathematical formulation of the resulting optimization problem, and application examples. Sections 10 and 11 discuss the potential weaknesses of our work, as well as the alleged generalizability of our approach. Finally, Section 12 summarizes our main contributions and the opportunities we envision for future research.

2. Related work

The literature on the *definition of requirements quality* is extensive. A recent systematic literature review on quality criteria for requirements can be found in [Heck & Zaidman 2018], where the authors summarize 28 different quality criteria for agile requirements specifications and compare them with those from traditional requirements engineering. Our work is based on our own previous systematization of quality criteria in three hierarchical levels of qualitative desirable properties (see the Introduction) and 18 quantitative measurable indicators (metrics) shortly described in Appendix I. See an extensive description of both criteria and metrics in [Génova et al. 2013] and [Parra et al. 2015]. This hierarchical arrangement has been the ground for the implementation of the Requirements Quality Analyzer (RQA) [Reuse Company 2016], which is the tool we use to extract the requirements metrics that feed, first, the learning algorithms, and second, the automatic rule-based classifier that assess the quality of new requirements.

Recent research on *automation in requirements quality* (still without artificial intelligence techniques) has progressed in different directions. Some works have been focused on the detection of ambiguities, inconsistencies and conflicts [Chantree 2006, Popescu et al. 2008, Kiyavitskaya et al. 2008, De Sousa et al. 2010, Wang et al. 2013, Sardinha et al. 2013, Ali et al. 2013, Aceituna et al. 2014]. Other works have developed algorithmic methods to measure the quality of more structured requirements in the shape of user stories [Lucassen et al. 2016]. There have been also projects aimed at the classification of requirements by topic, in order to assist reviewers in the evaluation of consistency and completeness of requirements [Ko et al. 2007, Ott 2013]. The detection of forward references to not yet defined terms has also been used as a measure of both individual quality of single requirements and global quality of requirements documents [Siahaan & Umami 2011]. Similar use of textual patterns, even if not directly focused on quality, has been applied to the classification of functional and non-functional requirements [Cleland-Huang et al. 2006, Cleland-Huang et al. 2007, Hussain et al. 2008]. Other studies have developed quantitative methods to evaluate the quality of requirements, with the goal of obtaining a prioritization of requirements that demand improvement [Fabbrini et al. 2001, Bucchiarone et al. 2005, Berry et al. 2006, Kasser et al. 2006, Otero et al. 2010, Génova et al. 2013, Thakurta 2013, Thitisathienkul & Prompoon 2015]. In general, however, all these methods are not tailorable to different contexts (see Section 4) or they are limited in the way they combine the different metrics (see Section 5); therefore, we think our method is really novel in this respect.

Artificial intelligence techniques, and specifically *machine learning algorithms* to emulate human judgment, have not been so extensively applied in this field. With a similar goal, but purporting the use of different techniques, there was a proposal (without implementation) to use neural networks and case-based reasoning to improve the quality of requirements [Jani & Islam 2012]. More akin to our research is the application of machine learning techniques to build a classifier that detects ambiguities in textual requirements [Hussain et al. 2007], a very good work that, to our knowledge, has had however no direct continuation; moreover, their focus was only on the ambiguous expression of requirements, whilst ours considers many more aspects of quality, such as the use of domain vocabulary, the size of requirements, the structure of sentences, and so on. The assurance of testability in non-functional requirements [Rashwan 2015] is another particular concern of requirements quality that has been targeted with machine learning techniques (in this case, employing support vector machines).

Other works have applied machine learning to detect software defects under the influential Orthogonal Defect Classification (ODC) framework, developed initially at IBM for software defect classification and analysis [Huang et al. 2015]; this work is indeed related to the quality of software, but not of textual requirements. Less related to quality, but still using machine learning, is a research aimed at the identification and annotation of requirements in user-generated content [Dollmann & Geierhos 2016]. Unsupervised pattern-based machine learning has also been used to determine requirements clusters for optimal definition of software development sprints in the context of agile software development [Belsis et al. 2014].

Search-Based Software Engineering [Harman & Jones 2001, Harman et al. 2012] is very apt for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and machine learning, among others. It has been successfully applied to solve, for example, the Multi-Objective Next Release Problem (MONRP) [Zhang et al. 2007], which is a good example of a Feature Subset Selection search problem.

3. Research Methodology

As it is well known, in the last decades of the 20th century a growing conviction consolidated: the scientific method developed for studying and analyzing natural phenomena was not apt to understand the design and construction of human artifacts, i.e. the products of engineering and technology [Génova et al. 2012]. The required method to produce an artifact should not start with the observation of phenomena, but rather with the *identification of a need, followed by artifact construction and evaluation* [Hevner et al. 2004]. This emerging field of construction-oriented research was called *design science*, the scientific study of design, and it was based on two assumptions: first, the design of artifacts can be a sophisticated task that contributes to the development of scientific knowledge; second, the scientific design of artifacts requires a specific research method [Frank 2006].

According to these guidelines, we succinctly describe now the research methodology we have followed in this work, with references to the sections of the paper where we deal more in-depth each aspect of the methodology. We also enumerate some basic assumptions in our research.

Identification of a need: problem statement and explicitation of goals. *Problem: the perceived quality of requirements is not universal, but highly subjective and context-dependent.* The demanded quality is not the same in large projects as in small ones, in safety-critical transportation or medical systems as in sheer news-displaying systems. Therefore, measuring quality based on universal rules is not enough; it should be based, instead, on the experts' interpretation of demanded quality in each different context, taking into account the needs and peculiarities of each project and organization.

However, since the involvement of domain experts is very costly and, at the same time, the automatic measurement of quality has demonstrated its partial success, we propose in this project the following *Goal: to develop a tool to automatically emulate the experts' judgment on requirements quality and automatically provide recommendations to improve it.* In order to achieve a quality assessment that is really tailored to a particular context, the emulation is performed based on previous expert quality judgments in that same context and organization.

The problem and the goals are described more in-depth in Sections 4 and 5.

Artifact construction. The constructive part of this project produces two different artifacts: first, an automatic classifier of requirements according to their perceived level of quality (see Sections 6, 7 and 8); second, an automatic recommender of modifications to improve the quality of a requirement that has been judged deficient (see Section 9). A high-level description of the construction process is also contained in Section 4.

The *automatic classifier* operates on objective metrics obtained from the requirements. Each metric constitutes a dimension in a multidimensional hyperspace of metrics, where regions of good and bad quality can be identified (see Section 5), i.e. certain combination of metrics values that can be associated with a given quality level. The classifier consists in a quality function that discriminates those regions by computing different combinations of metrics that correspond to each region.

Instead of defining an *a priori* quality function (e.g. based on universal definitions of quality), the procedure employs artificial intelligence techniques (machine learning) to learn and emulate the experts' quality judgment, and build a quality function that is tailored to the particular context of application (see Section 6). This requires a previous training data set of requirements, extracted from a particular domain with its particular level of demanded quality, which the experts have manually classified according to their perceived quality.

The *automatic recommender* operates on the same objective metrics obtained from the requirements, and it applies search-based software engineering principles to find a list of modifications, sorted according to the required effort to achieve them, so that the user can choose one that entails a real improvement and the defective requirement, once amended, can satisfy the quality function that emulates expert judgment (and presumably also the experts themselves).

Artifact evaluation. The *automatic classifier* has been evaluated through stratified 10-fold cross validation, which is a standard procedure in the field of machine learning. Specifically, the effectiveness of the classification is measured as its *accuracy*, which is a standard performance metric consisting in the percentage of agreement between the experts' classification and the automatic classifier, i.e. the ratio of true positives plus true negatives to all existing training instances. The initial data set we have used in our experiments is a corpus of 1035 textual software requirements from the domain of aerospace industry, together with their quality classification, provided by experts of the INCOSE's Requirements Working Group. The classifier obtained reaches more than 85% in accuracy, which qualifies it as reasonably good in order to provide useful advice to requirements authors. When the training set is imbalanced, other metrics such as precision, recall and F-measure are more meaningful than accuracy; therefore, even if our data set is very well balanced, we define all these metrics (see Section 6) and provide their results for a more complete picture (see Section 7).

The *automatic recommender*, in contrast, has not undergone a full-blown evaluation of its effectiveness. However, we consider that it is useful to present the recommender as a complement to the classifier, since it offers concrete suggestions to achieve the desired quality for defective requirements (see Sections 8 and 9).

Basic assumptions. We think it is useful to enumerate some basic assumptions in our research. Checking that they are realistic assumptions strengthens our position (some may look too evident to some readers, but not to others):

- Requirements are written in natural language.
- Requirements are stored in electronic format, in individualized units.
- The text of the requirements is electronically processable.
- Requirements should adhere to recognized style guides and standards; departing from this rule is considered a defect.
- Requirements must use the domain vocabulary; departing from this rule is considered a defect.
- There exists a definition for the domain vocabulary, in the form of an ontology.
- There exists a tool (in our case, RQA, but it could be a different one) by means of which we can extract a certain number N of objective requirements metrics, so that each requirement can be represented as a vector in an N-dimensional hyper-space.
- Points representing requirements with similar quality will be closely situated in this hyperspace, forming more or less compact clusters or regions in the cloud of requirements; these groupings form the basis for the extraction of patterns of good or bad quality.

4. Motivation: flexible assessment of requirements quality

Measuring and classifying is the first step towards automation. As we have stated in a previous work, we can measure certain objective metrics on textual requirements, and through them we can obtain the desired evaluation of requirements quality [Génova et al. 2013]. However, the perceived quality of requirements is not universal, but highly subjective and context-dependent. For example, if we consider the typical distinctions between user requirements and software requirements [ESA 1995], it is clear that they demand different writing styles: *user requirements*, which are client-oriented, must be perfectly understandable to stakeholders and solution-independent; *software requirements*, instead, may be allowed to mention terms that, from a user's viewpoint, are too close to design. Therefore, performing automatic measures based on fixed policies is not enough.

One of the most decisive factors for a good assessment of requirements quality must be the involvement of domain experts, since their personal (subjective) evaluation is strongly linked to the needs and peculiarities of the project and the organization. Today, the most widely used techniques for the analysis of requirements that automatically compute quality metrics do not take into account the experts' interpretation of quality and quality levels of requirements *in each different context*, but rather rely on general rules of quality. Besides, these techniques are not tailorable, they are not flexible and adaptable to different projects and organizations; they are, therefore, rather limited.

In this work we present a method for the evaluation of the quality of requirements in an automatic way, according to the quality rules and criteria employed, more or less implicitly, by the domain experts in the organization, without need of a previous and explicit definition of those criteria, which besides could be a very costly task. The objective of this method is to *emulate the experts' judgment* on the quality of new requirements that are entered in the system. In order to achieve this goal, the experts must contribute with an initial set of requirements that they have previously classified according to their quality, and that they have chosen as appropriate for establishing the demanded standard quality (this implies that the initial set must include requirements classified in all quality levels; in other words, including only good requirements is not enough). For each of the requirements in the given set, we extract metrics that quantify the various dimensions of quality already presented in previous works [Génova et al. 2013, Parra et al. 2015].

We then use machine learning techniques (namely Rule Inference) to emulate the implicit expert's quality function, i.e. the value ranges for the metrics, as well as the way the metrics are combined, to yield the interpretation of requirements quality by the domain expert. The result will be a computable formula made of simple arithmetic and logical operations. The advantage of using a combination of rules, in contrast to neural networks and other techniques [Major & Mangano 1995], is that, even if the complete formula need not be simple, each component rule is easily interpretable by the experts and other users, so that the rule can be edited if necessary.

Obviously, the expert does not apply a computable formula to judge on the quality of requirements. Moreover, the expert's judgment can be based on metrics that are different from those we use, or not based on metrics at all. However, we assume (this is our working hypothesis) that we can learn and emulate, at least to a certain degree, the expert's 'educated taste for quality' [Génova & González 2016] with our method. We don't expect machine learning will be apt to tell us what 'Quality' is, which metrics are adequate and representative of requirements quality, or how to measure them. But we do think that, once those metrics have been proposed and defined, machine learning can help us to tell whether and how they are related to the expert's judgment, and what computable formula has the best fit.

More formally, our hypothesis is that an automatic rule-based classifier obtained through machine learning algorithms, fed with training data extracted from a particular domain and a particular level of demanded quality, can emulate the expert's judgment on requirements quality, with a level of effectiveness enough to provide useful advice to requirements authors. The details of the hypothesis are explained in Section 6, and the experimental results that support our hypothesis (more than 85% in percentage of agreement) are presented in Section 7. See also the discussion about the goodness of these results in Section 10.

This method has the advantage of being tailorable to different situations, different domains, different styles to write requirements, and different demands in quality. In order to achieve this, we need a tool that computes quality metrics on textual requirements, and the initial set of requirements previously classified by the expert, so that we can feed the learning algorithms. Of course, the method requires a significant investment in order to obtain and tune the quality metrics tool, and to obtain a sufficiently large set of labeled requirements pertaining to the domain, so that machine learning algorithms can be assured to produce useful and trustworthy automatic classifiers.

The main contribution of our work, then, is a method to build a classifier that, using conventional machine learning techniques, learns from the information provided by the expert of that particular organization, and that adapts itself to best emulate the expert's judgment; above all, the method generates human-readable rules from the expert's tacit knowledge, which can be reworked and improved. Besides, we can provide automatic suggestions to improve the requirements, by computing the quality rules that could be satisfied to change a requirement from Bad to Good. We think the method will be useful in medium/large projects and organizations; small organizations, in contrast, will not benefit from it, since the payoff is probably not worth the necessary investment.

The stakeholders that will directly benefit from the method are *requirements authors*, who will have a new tool at hand to achieve, with less effort, their own goal of improving requirements quality; their *supervisors* and the *clients* of the system are also indirect stakeholders that will benefit from any improvement in the quality of requirements. Requirements authors will have a tool to automatically assess the quality of the requirements they are writing, based on expert judgment adapted to the concrete working context; the tool will also provide concrete recommendations to improve the quality of requirements that are found to be deficient.

For the demonstration of the reliability of the proposed method, we examine a set of requirements provided by the INCOSE (International Council on Systems Engineering). This set of requirements is protected by confidentiality, but we provide a table [Moreno et al. 2016] with the metrics we extracted with the RQA tool, *Requirements Quality Analyzer* [Reuse Company 2016], together with the classification given by the experts, without disclosing the text of the requirements themselves. In this case the classification of quality is binary (Good, Bad), not because we demanded it to be so, but because it was what the experts provided. In any case, the method is independent of the number of quality levels: it will yield as many levels as they are present in the initial set.

5. The problem: quality functions in the hyperspace of metrics

Since the quality of requirements is multi-facetted, that is, it consists of different properties that are not directly correlated to each other (e.g. completeness, consistency, correctness), then measuring and improving quality by combining different metrics becomes a *multi-objective problem* that usually cannot be solved with the simplest methods (i.e. linear combinations of metrics).

Suppose we classify requirements by their quality based on a single variable or metric, for example the 'number of domain terms' (NDT) used in the requirement. We can then formulate a simple rule to *transform the metric into a quality level* [Génova et al. 2013], such as "if NDT = 0 then Bad, else Good". The rule can be easily refined to account for more quality levels (Bad, Dubious, Good), using more intervals in the value of the metric, such as "if NDT <= 0 then Bad, else if NDT > 4 then Dubious, else Good" (see Figure 1). For simplicity, since the argument is easily generalizable to whatever number of quality levels, we will assume in the rest of the paper that the number of quality levels is only two (Bad, Good), whilst the number of metric intervals is open: "if NDT <= 0 then Bad, else if NDT > 4 then Bad, else Good".

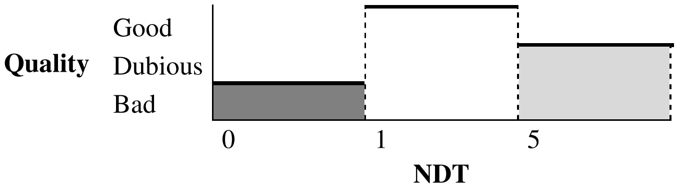

Fig. 1 A simple rule to transform a metric such as the number of domain terms (NDT) into a quality level

Now, suppose we want to combine two different metrics to assign a quality level, such as NDT and 'size in words' (SW). We can represent both variables on the X-Y plane, where each point is a requirement and the color is the quality level. The simplest way to discriminate quality within the cloud of points is by a linear combination of the variables, i.e. the traditional method of weighted average of metrics: “if (a*NDT + b*SW) <= L, then Bad, else Good”, where a, b and L are convenient values. When the cloud of points is naturally split in two regions of quality separated by a straight line, the values of a, b and L can be easily obtained with simple mathematical methods (see Figure 2). The method generalizes to any number of dimensions (metrics) that define a hyperspace of requirements split in two regions by a hyperplane.

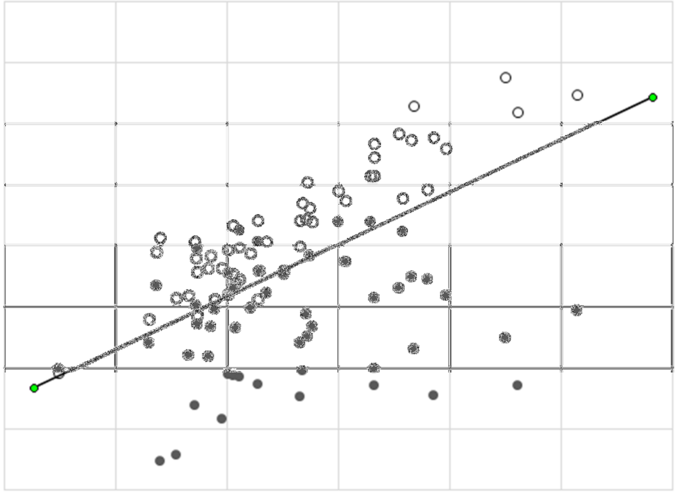

Fig. 2 Combination of two arbitrary metrics using a straight line to discriminate quality in the simplest case (white: Good quality, black: Bad quality)

However, this simplicity is not usually the case when the variables present a more complex relationship with quality that can be very difficult to estimate *a priori* (see Figure 3). In these situations, a more convenient, yet simple way to combine the metrics is by means of a rectangular region: quality is Good inside the region, otherwise it is Bad. Note that the region could be open in one or more sides: “if NDT between [1, 4] and SW between [10, –] then Good, else Bad”. The method is also rather simple and generalizes to hyper-rectangular regions in the hyperspace of requirements.

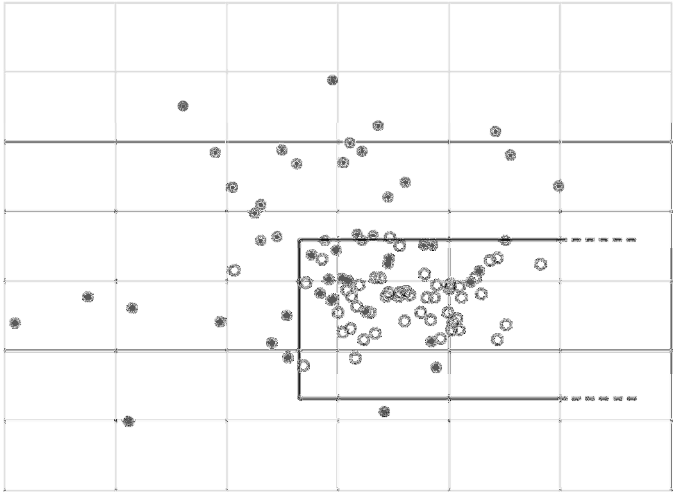

Fig. 3 Combination of two metrics using a single (open) rectangle to discriminate between Good (white) and Bad (black) quality

The most usual case in the combination of metrics, however, is not at all so simple, and the cloud of points is not easily discriminated by a single (hyper-) rectangle. In this situations we can still generalize the procedure to a combination of regions that can be better adjusted to the cloud of points (see Figure 4), such as: “if NDT between [1, 3] and SW between [10, 30] then Good, else if NDT between [4, 7] and SW between [20, –] then Good, else Bad”. Note that the regions could be overlapping (requiring that they do not overlap could produce a worse adjustment of the rule, or a more complex rule with more rectangles).

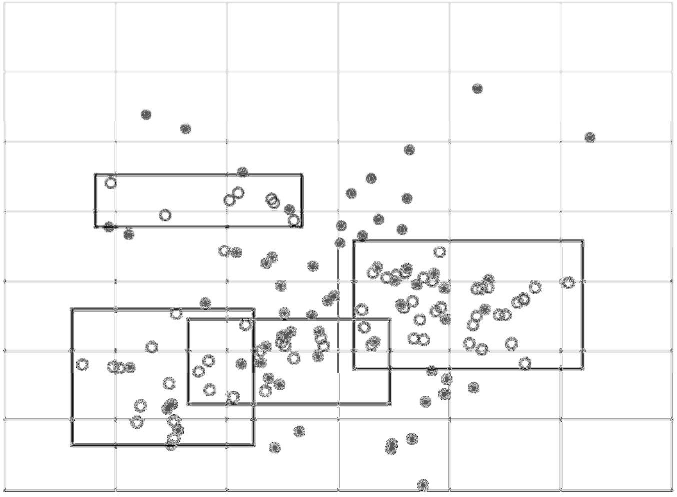

Fig. 4 Combination of two metrics using a combination of rectangular regions to discriminate between Good (white) and Bad (black) quality

Summing up, the discrimination of regions by means of hyperplanes (Figure 2) is generally very inadequate, and the use of simple rectangular regions (Figure 3) is still insufficient. Instead, a combination of rectangular regions (Figure 4) is a versatile method when the different metrics employed present complex relationships manifested in the clustering of points. The higher the number of regions, the better the adjustment of the discriminating rule to the data set. However, computing the concrete intervals for each metric, i.e. the sizes of hyper-regions, becomes a difficult problem. This is where machine learning techniques prove particularly useful to generate the rules.

6. Method: machine learning techniques

The discrimination of requirements quality is achieved in this project by an automatic classifier that has been trained by means of machine learning in order to identify those hyper-regions of similar quality that have been explained in the previous section. Machine learning is a well-known subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence, therefore we provide only a brief explanation. Machine learning explores the construction and study of algorithms that can learn from data, by *building a model from example inputs* in order to make data-driven predictions or decisions, rather than following strictly static program instructions [Bishop 2006]. On the other hand, Search-Based Software Engineering is an approach of Software Engineering in which search-based optimization is applied to software engineering [Harman & Jones 2001]; machine learning, then, is one of the techniques that can be used to perform search-based software engineering.

Machine learning techniques can be supervised or unsupervised [Russell & Norvig 2003, Weiss & Indurkhya 1998]:

- *Supervised learning:* the algorithm is presented with example inputs and their desired outputs, and the goal is to learn a general function that maps inputs to outputs.
- *Unsupervised learning:* the inputs to the learning algorithm are given no output labels, leaving the algorithm on its own to find a structure or pattern in its input.

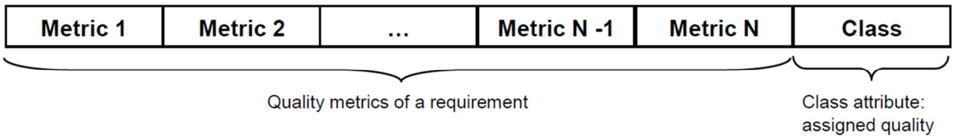
Metric 1
Metric 2
...
Metric N -1
Metric N
Class
Quality metrics of a requirement
Class attribute: assigned quality

Fig. 5 Format of the training instances used as input to the learning algorithm. Testing instances have the same format

More specifically, supervised learning is the task of inferring a function from labeled training data, where each input is described by a vector of common attributes (see Figure 5). The *training data* consist of a set of training examples, each example consisting of an input object (in our case, the vector of quality metrics obtained from each textual requirement) and a desired output value (in our case, the requirement's quality as judged by the expert). A supervised learning algorithm analyzes the training data and infers a generalized function, which can be used for classifying new inputs. The effectiveness of the automatic classifier obtained is then measured against the *testing set*, which has been previously segregated from the initial data, so that training and testing are performed with different data sets.

The effectiveness (or performance) of the classification can be measured in different ways, each standard measurement having different properties. Let DS be the absolute number of instances in the data set; TP the number of 'true positives', TN the number of 'true negatives', FP the number of 'false positives', and FN the number of 'false negatives'. Obviously, $DS = TP + TN + FP + FN$. Then:

- **Accuracy** $A = (TP+TN)/DS$ is the ratio of true positive classifications plus true negative classifications to all existing instances, i.e. the percentage of agreement between the automatic classifier and the experts' input classification.
- **Precision** for positives $P_p = TP/(TP+FP)$ is the ratio of true positive classifications to all positive classifications, i.e. the percentage of positive instances correctly classified as positive. Respectively, Precision for negatives is $P_n = TN/(TN+FN)$.
- **Recall** for positives $R_p = TP/(TP+FN)$ is the ratio of true positive classifications to all relevant instances, i.e. the percentage of relevant instances correctly classified as positive. Respectively, Recall for negatives is $R_n = TN/(TN+FP)$.
- **F-Measure** for positives $F_p = 2/(1/P_p+1/R_p)$ is the harmonic mean of Precision and Recall for positives, and F-Measure for negatives is $F_n = 2/(1/P_n+1/R_n)$.

Accuracy is the most intuitive measurement, but it is also the weakest one when the data set is imbalanced (a naïve classifier giving *always* the answer 'true' would be perfectly useless, even if it would have a 99% accuracy in a data set with 99% positives and 1% negatives). Precision is better against false positives, and Recall is better against false negatives (respectively Precision and Recall for negatives). F-Measure considers both at the same time. In the next section we give the results of all these measurements.

In terms of search-based software engineering, the optimization problem consists in *finding a function of the quality metrics* (a piecewise function defined by a set of rules), such that it minimizes the distance with the experts' quality evaluation over the set of all requirements. This can receive the following mathematical formulation:

- Let R be a set of textual requirements, $R = \{r_1, \dots, r_n\}$.
- Let C be the quality classifications over the set of requirements, $C = \{c_1, \dots, c_n\}$, such that $c_i \in \{0, 1\}$ is the quality provided by the experts to requirement r_i, where 0 represents bad quality and 1 represents good quality.
- Let M be a set of correctness metrics applied to requirements, $M = \{m_1, \dots, m_k\}$, such that $m_j\colon R \rightarrow \mathbb{R}$ $(1 \leq j \leq k)$.

Goal:

Find a function $f\colon \mathbb{R}^k \rightarrow \{0, 1\}$ such that it minimizes

$$\sum_{i=1}^{n} | f\big(m_1(r_i), \dots, m_k(r_i)\big) - c_i |$$

Among various machine learning techniques, rule induction (or *rule inference*) is a kind of supervised learning that process input training data and *produce a set of IF-THEN rules used to classify the new examples* [Clark & Niblett 1989, Hong et al. 1986]. Two main strategies are commonly used:

- Produce a decision tree and then extract its rules.
- Generate the rules covering all the examples in a given class; exclude the covered examples and proceed with the next given class, until all classes are covered.

The first strategy is implemented in the C4.5 system [Quinlan 1993], which extends the previous ID3 [Quinlan 1986]. Other algorithms such as PRISM [Cendrowska 1987] are based only on covering, whilst PART [Frank & Witten 1998] combines both strategies.

These strategies present the following advantages to minimize the impact of unintentional errors in the expert's classification of the quality of requirements used as training examples:

- Robustness against noise due to errors, omissions or insufficient data.
- Identification of irrelevant attributes.
- Detection of absent attributes or knowledge gaps.
- Extraction of expressive and easy to understand rules.

- Possibility to interpret or modify the produced rules with aid of expert knowledge, or even to incorporate new rules inferred by the experts themselves [Major & Mangano 1995].

In order to improve the effectiveness of the individual classifiers obtained by means of rule induction, *ensemble methods* construct a set of classifiers instead of a single one, and then classify new instances by taking a vote of their decisions (it can be a weighted vote, the mode of the votes, etc.) [Dietterich 1997]. The technique has two main variants:

- *Homogeneous classifiers* are generated with the same learning algorithm [Dietterich 2000]. The main methods are Bagging [Breiman 1996] and Boosting [Schapire 1990].
- *Heterogeneous classifiers,* instead, are generated with different learning algorithms. The most used method is Stacking (Stacked Generalization) [Wolpert 1992].

We performed experiments using the C4.5 and the PART algorithms, as well as these two algorithms enhanced through homogeneous classifiers (Bagging and Boosting). We explain the experiments in detail in the next section.

Summing up, the whole process consists of two main stages: rule inference and rule application. The *inference stage* can be summarized in the following steps (see Figure 6):

1. Obtain the initial set of textual requirements.
2. Classify requirements according to their quality, as judged by human experts.
3. Extract quality metrics by means of an automated tool, in our case the RQA tool.
4. Build the training and testing data for the supervised machine learning algorithm, combining the two previous outputs: each requirement is represented as a vector of quality metrics and human-judged quality level (see Figure 5).
5. Launch the rule inference learning algorithm (in our case, run in the Weka tool) to obtain as output the automatic classifier, i.e. the function made of rules that maps requirements to quality levels, thus emulating the human experts; this step includes a standard validation of the classifier through testing data, providing the different effectiveness metrics mentioned before (accuracy, precision, recall, and F-measure).

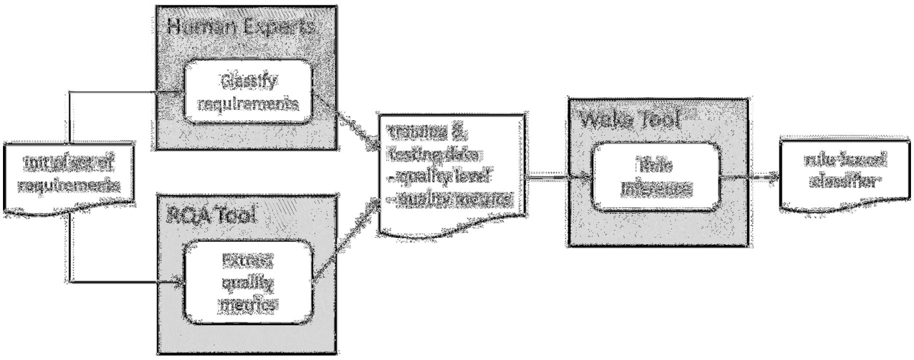

Fig. 6 Inference stage: obtaining the expert's implicit quality function through machine learning

Once we have built a validated rule-based classifier, we can use it as input in the *application stage* to automatically classify new requirements as the emulated human experts would do (see Figure 7):

6. Obtain a new set of textual requirements and extract their quality metrics.
7. Classify the requirements using both the definition of the automatic rule-based classifier and the new requirements metrics as input to the Weka tool.

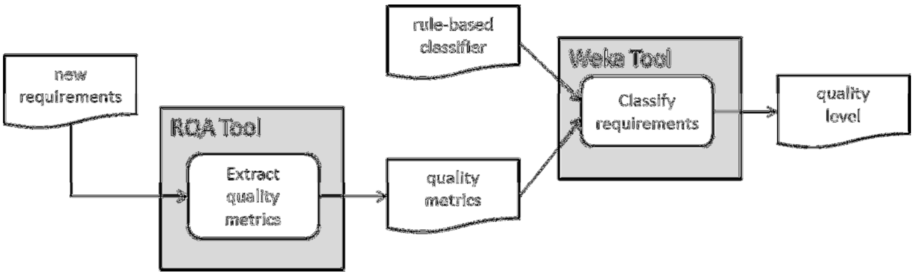

Fig. 7 Application stage: emulating the expert's judgment with the generated rule-based classifier

Note that the classification of requirements is now automatic, which is the whole point of this research. But this classification is performed according to rules that have been extracted from training examples provided by experts. If the training examples were different, the automatic classification would be different, too. This is what makes the whole process tailorable to different contexts, different styles to write requirements, and different demands in quality, without need that the experts explicitly formulate their quality rules, and saving the effort required to do that. There is no gold quality standard, no universal rules that can be applied in every context.

7. Description of the experiments and results in effectiveness and efficiency

The initial data set we have used in our experiments is a corpus of 1035 textual software requirements from the domain of aerospace industry, together with their quality classification, provided by experts of the INCOSE's Requirements Working Group. These are experienced requirements engineers and researchers, both from academy and industry, whose purpose is to advance the state of the practices, education and theory of requirements engineering and its relationship to other systems engineering functions [INCOSE 2012]. The requirements were originally classified by these experts in two well-balanced levels: 545 requirements with good quality and 490 with bad quality.

Next we automatically extracted quality metrics from the requirements with the RQA tool [Reuse Company 2016], using its standard out-of-the-box configuration. The set of 18 quality metrics used in this project has been extensively described and justified in [Génova et al. 2013] and [Parra et al. 2015], so we give only a short description in Appendix I. As we mentioned before, the text of the requirements is confidential, therefore we provide a table [Moreno et al. 2016] only with the extracted metrics and the original quality classification.

Based on this data set, we launched the six different learning algorithms (C4.5 and PART, in three variants each) to build the corresponding automatic classifiers. In order to estimate their effectiveness, we performed, as it is usual, a stratified 10-fold cross validation [Kohavi 1995]. This means that the whole sample is randomly partitioned into 10 equal sized subsamples (folds) with the same proportion of good and bad requirements. Each fold is then used as a testing set, with the remaining 9 folds used as training set. The cross validation process is repeated 10 times, so that 10 different testing classifiers are obtained, with each of the 10 subsamples used exactly once as validation data. This standard method has the advantage that all instances are used for both training and testing, and each instance is used for validation exactly once. In other words, for each learning algorithm, a final classifier is obtained with training based on the whole data set, and its effectiveness is estimated through the *average effectiveness* of the 10 testing classifiers obtained with the 10-fold cross validation method.

The experiments (generation of classifiers and estimation of effectiveness) were implemented in the popular Weka suite [Witten & Frank 2000], version 3.6.12, keeping its standard default parameter configuration. We show in Table 1 the results obtained in the effectiveness of the classification of both C4.5 and PART, as well as the enhanced versions of these two algorithms by means of the bagging and boosting techniques. As we have mentioned before, we give four different metrics of effectiveness (defined in the previous section): accuracy, precision, recall, and F-measure, respectively for positives (i.e. instances classified as Good) and for negatives (classified as Bad).

Table 1 Effectiveness of the classifiers obtained with each algorithm, estimated through 10-fold cross validation (average accuracy, precision, recall and F-measure, respectively for positives on the left and negatives on the right)

Algorithm	**C4.5**		**PART**		**Bagging C4.5**		**Bagging PART**		**Boosting C4.5**		**Boosting PART**	
Average Accuracy	82.51		85.31		85.12		86.18		87.25		86.57	
Average Precision	83.2	81.7	85.7	84.9	85.2	85.0	86.8	85.5	86.9	87.6	86.8	86.3
Average Recall	83.7	81.2	86.6	83.9	86.8	83.3	87.0	85.3	89.2	85.1	87.9	85.1
Average F-Measure	83.4	81.5	86.1	84.4	86.0	84.1	86.9	85.4	88.0	86.3	87.3	85.7

As it can be observed, PART is generally better than C4.5, Boosting is better than Bagging, and Bagging is better than the base algorithms without ensemble enhancement. Boosting C4.5 is finally the most effective algorithm in our experiments, peaking to 88% in the F-measure that combines precision and recall.

Regarding the efficiency of the algorithms, Table 2 shows the time required by the rule learning process with the generation algorithms using the Weka suite. The algorithms have been executed on a computer with Windows 10 operative system on top of an Intel microprocessor Core i7-4770 to 3.40 GHz and a RAM capacity of 16 GB. These values are interesting because of the necessity to regenerate the classifiers in new contexts (new projects, new quality constraints on the requirements, new computed values of the metrics, etc.). It can be noted that the enhancement by means of Bagging and Boosting requires more time than the base algorithms C4.5 and PART alone, as it would be expected. In any case, the differences are not so relevant, since *the* *regeneration of the rules will not be a frequent task*. In fact, these generation times can be considered practically negligible, but it is important that we know that.

Table 2 Efficiency of generation: time in seconds reported by Weka to generate the classifiers with the different learning algorithms

Algorithm	C4.5	PART	Bagging C4.5	Bagging PART	Boosting C4.5	Boosting PART
Generation Time (s)	0.17	0.09	0.28	0.84	0.32	0.90

8. Analysis of the obtained rule-based classifiers

The output of the learning algorithms has the appearance shown in Figures 8 and 9. C4.5 produces a *decision tree*, where branches express conditions, and leaves represent final decisions. In this example, the first leaf tells that the requirement is classified as Bad if it contains no design sentences and no domain verbs. This rule classifies 166 instances as Bad, of which 4 instances have been wrongly classified, according to the original classification provided by the experts (i.e. the rule gives its precision in classifying instances; in this case, the first leaf has a precision of (166-4)/166=0.976). If that leaf is not reached, the evaluation proceeds down the decision tree. The classifier generated with our data produce a tree with 71 leaves (35 resulting Good, 36 resulting Bad).

```
Design_sentences <= 0
|   Domain_verbs <= 0: Bad (166/4)
|   Domain_verbs > 0
|   |   Flow_sentences <= 0
|   |   |   Domain_verbs <= 1: Bad (91/6)
|   |   |   Domain_verbs > 1
|   |   |   |   Readability <= 7: Good (7/0)
|   |   |   |   Readability > 7
|   |   |   |   |   Conditional_mode <= 0
|   |   |   |   |   |   Domain_verbs <= 2: Bad (37/2)
|   |   |   |   |   |   Domain_verbs > 2
|   |   |   |   |   |   |   Readability <= 10: Good (3/0)
|   |   |   |   |   |   |   Readability > 10: Bad (6/1)
|   |   |   |   |   Conditional_mode > 0: Good (3/0)
|   |   Flow_sentences > 0: Good (8/0)
...
...
```

Fig. 8 Appearance of the first lines of the output of the C4.5 rule learning algorithm

On the other hand, PART produces a *decision list* made of rules with precedence. The first rule in the example tells that the requirement is classified as Bad if it contains no design sentences, no domain verbs, and no connectors. Again, this rule classifies 142 instances as Bad, of which 1 instance has been wrongly classified. If the rule is not satisfied, the evaluation proceeds with the next rule. The classifier generated with our data produce a list of 54 rules (27 resulting Good, 27 resulting Bad).

```
Design_sentences <= 0 AND
Domain_verbs <= 0 AND
Connectors <= 0: Bad (142/1)

Design_sentences <= 0 AND
Flow_sentences <= 0 AND
Text_length_(words) > 30: Bad (56/0)
…
…
```

Fig. 9 Appearance of the first lines of the output of the PART rule learning algorithm

Reaching a leaf in the first case (decision tree) or satisfying a rule in the second case (decision list) means the requirement, as represented by its metrics vector, has been enclosed within a region of defined quality (Good or Bad) in the hyperspace of requirements (see Section 5). When the application of the rules proceeds on, the rule that is finally satisfied, and the definition of the corresponding hyper-region, is more and more complex, due to the precedence of the previous rules. For example, the satisfaction of the second rule in Figure 9 implies certain intervals of values not only for the metrics Design_sentences, Flow_sentences and Text_length_(words), but also for Domain_verbs and Connectors in the first rule.

As we have mentioned above (see Section 6), one of the advantages of rule induction algorithms is that they are able to discard irrelevant attributes, i.e. metrics that in fact are not necessary to emulate the experts' judgment. In our case, the finally obtained classifiers do not use the metrics Rationale_sentences, i.e. they use only 17 from the 18 metrics computed by the RQA tool in this project.

Now, one of the most interesting and practical aspects of the obtained rule-based classifiers is that, for a given requirement that has been classified as Bad, we can find the list of rules that can be satisfied to improve the requirement. In other words, we can offer a list of recommendations to modify a bad requirement so that it becomes Good. We explain the mathematical details of this problem in the next section.

9. Getting recommendations to improve requirements

Similarly to the problem of finding a function that emulates the experts' quality classification of requirements, we can formulate the optimization problem of finding a least-effort modification of a requirement that changes its class from Bad to Good. The best modification is the one that most improves the quality of the requirement, not necessarily the least costly. However, in the context of a binary classification any change of class (from Bad to Good) implies the same increment of quality. Our approach, then, is to provide the requirements author with a list of suggested modifications, sorted according to the required effort to achieve them, so that the user can choose one that entails a real improvement and at the same time is relatively easy to achieve.

In terms of search-based software engineering, the mathematical formulation is as follows:

- Let C be an automatic rule-based binary classifier that decides on the quality of a requirement. The decision is taken on a set of N numerical attributes of the requirement, previously computed with a quality analyzer tool, in our case RQA.

- Let R be a requirement that has been classified by C with Bad quality.
- Let M_i, $i\in\{1..N\}$, be the values of the metrics obtained for R, based on which C has classified R as Bad. R can be represented also as the vector (M_1, M_2, …, M_N).
- Let P_j, $j\in\{1..L\}$, be the antecedents of the positive rules in C, i.e. those rules that classify a requirement as Good. Then we have, $\forall j\in\{1..L\}$, $\neg P_j(R)$, or equivalently $\neg P_j(M_1, M_2, \ldots, M_N)$. In other words, all antecedents of positive rules, when applied to the vector of metrics of a bad requirement, classify it as Bad.

Suppose now we want to rewrite R into R', so that R' is evaluated as Good; that is, R' is such that it satisfies the antecedent of some positive rule: $\exists j\in\{1..L\}$, $P_j(R')$, or equivalently $\exists j\in\{1..L\}$, $P_j(M'_1, M'_2, \ldots, M'_N)$.

Besides, we want to minimize the effort to obtain R'. Suppose there is a fixed cost to modify the value of a metric, and a cost dependent on the magnitude of the modification. Let $F_i\in \mathbb{R}^+\cup\{0\}$, $i\in\{1..N\}$, be the estimated fixed cost to modify the value of M_i; and let $K_i\in \mathbb{R}^+\cup\{0\}$, $i\in\{1..N\}$, be the estimated cost to increase or decrease by an amount of 1 the value of M_i.

Then the cost to modify R so that M_i becomes M'_i is $F_i + K_i\cdot|M_i-M'_i|$. We want to minimize the function $\Sigma^n_{i=1}\ F_i + K_i\cdot|M_i-M'_i|$.

Goal:

Find a modification R' of R so that

$$\exists j\in\{1..L\},\ P_j(M'_1, M'_2, \ldots, M'_N)$$

and with a minimum modification cost

$$\sum_{1\le i\le n\ \wedge\ M_i\neq M'_i}^{n} F_i + K_i\cdot|\,M_i - M'_i\,|$$

Note that, in the preceding formula, we add the fixed cost F_i only when the modification of the requirement affects the i-th metric, i.e. when $M_i \neq M'_i$. The formulation could also be generalized to consider that the cost to increase the value of a given metric is different from the cost to decrease it (for example, adding a domain term could be more costly than deleting one).

We have implemented an algorithm that solves this optimization problem in the case of C4.5 generated rules, with heuristic values for the F_i and K_i coefficients. The algorithm provides not only the absolute least-effort modification (R'), but a list of suggestions sorted according to their cost (R', R'', R'''…), so that the user can consider other relevant factors beyond cost to accept a suggestion. For example, we can use the precision and the number of classified instances for each rule generated by the learning algorithm (see the explanation of the decision tree in Figure 8) as another useful criterion to choose among the proposed list of modifications (a rule that classifies many instances with higher precision is better related to quality). Next we show an example application of the algorithm, with only the least-effort modification for simplicity.

Example:

Let's take the following requirement that has been classified as Bad by the C4.5 classifier obtained in Section 7:

Typically *the interface component* ***must*** *be able to* ***handle*** *at least one* ***fast connection*** *with the service component* ***and*** *one* ***connection*** *with the predefined* ***data base*** *component.*

This requirement has the following non-null metrics:

Paragraphs	1	
Words	27	
Readability	16	
Punctuation	171	
Connectors	1	and
Ambiguous	2	typically, fast
Design	1	data base
Imperative	1	must
Domain concepts	1	connection (twice)
Domain verbs	1	handle

We find that a least-effort modification can be achieved to satisfy the following rule, extracted from the decision tree:

```
Design_sentences > 0
|   Incomplete_sentences <= 0
|   |   Domain_verbs <= 1
|   |   |   Domain_concepts > 3: Good (17/1)
```

The requirement satisfies all the values in the rule, safe for domain concepts, which according to the rule should be strictly greater than 3 (remember that this rule has sense only within the context of all rules, i.e. as defining a hyper region among other hyper regions in the hyperspace of metrics). Therefore, the rule can be satisfied by the requirement with the modification of a single metric, by adding three domain concepts. Then we can provide the following recommendation: "increase the number of domain concepts by 3", so that the modified requirement could become:

Typically *the interface module* ***must*** *be able to* ***handle*** *at least one* ***fast connection*** *with the service module* ***and*** *one* ***connection*** *with the predefined data base module.*

We have replaced 'interface component', 'service component' and 'data base component' by their corresponding domain concepts in the ontology: 'interface module', 'service module' and 'data base module'. In fact, we could change the ontology instead of the requirement to make the fit. Once the requirement has been modified, we must extract the new metrics and check that its quality level is acceptable, as it is the case in this example.

Obviously, it is not simply a question of adding domain terms to satisfy the rule and deceive the tool; but it could happen that the requirements engineer could have inadvertently used wrong terms instead of the approved ones (‘interface component’ instead of ‘interface module’, say) and the tool can help discover those mistakes. Of course, the engineer should not blindly follow the recommendations of the tool. Since our algorithm produces a list of recommendations ordered by the effort required, if the first recommendation is not appropriate from a professional viewpoint, the engineer can take the next one, and so on. Similarly, in a general authoring context, a writing assistant (such as modern word processors offer) can tell some of the sentences in a text are too long: the recommendation can be responsibly followed or not, but the tool is giving useful advice anyway.

Note that the two ambiguous sentences detected (‘typically’, ‘fast’) need not be removed in order to satisfy this rule. What does this mean? We can interpret it as follows: the automatic rule-based classifier has learned from the expert that the lack of domain concepts is a severe defect, while the presence of ambiguous sentences is tolerable; then, the original version of this requirement is easier to amend by adding domain concepts than by removing ambiguous sentences (because the latter change would not be enough). Of course, a different expert could have put more emphasis on ambiguity-related defects, and that would have been reflected in a different machine-learning generated classifier. The point is, we do not intend to provide an ‘absolute’ quality assessment, but to emulate with flexibility the particular judgment of a given expert in a given context.

10. Potential risks and threats to validity

As it happens with every machine learning process, the effectiveness of the output assessment algorithm is highly dependent on its inputs. First, the selection of metrics, the reliability of the metrics computing tool, and its configuration. In this case we refer to our previous works [Génova et al. 2013, Parra et al. 2015] where the metrics have been justified and the industrial acceptance of the RQA tool [Reuse Company 2016] has been demonstrated.

Second, the initial classification by the experts (the set of training and testing instances) is also of capital importance. If the initial data set is wrong, the automatic classification of requirements by their quality will be useless. This factor is completely out of our control. However, its importance is relative: what we demonstrate is that we can emulate the experts’ judgment on the basis of (a) their personal (subjective) quality assessment and (b) objective, measurable attributes of the requirements; if the learned rules are “wrong”, it is the experts’ fault, not ours. The better the original classification, the better the automatic classification. This apparent weakness is in fact one of the strongest points in our method, since it allows us to build a flexible, context-dependent evaluation tool. For example, the vocabulary in software requirements might accept terms that are closer to the solution, in contrast to user requirements that would tag them as unacceptable design terms.

The experts themselves might be inconsistent in their quality assessment. For example, we have detected a disagreement of 10-20% among different experts working in the domain of photographic image quality [Robledano et al. 2016]. Even the same expert might give different evaluations in successive rounds on the same set of requirements. We consider this is a natural phenomenon, since the experts are not machines that behave always the same: they are people that are influenced by their experience (they are ‘experts’), and that can change their behavior accordingly as their experience increases [Génova et al. 2012]. In fact, reaching more than 85% in effectiveness qualifies our method as reasonably good. Remember this effectiveness has been estimated, as it is usual in machine learning, through four different metrics (accuracy, precision, recall and F-measure) with the 10-fold cross validation method. This means that the rule-based classifier can pretty well emulate the experts’ judgment, even if the experts do not use explicit rules to assess quality. The automatic assessment has still false positives and false negatives (around 15%), but it undoubtedly provides a fruitful ground to suggest improvements.

It could also happen that requirements with similar metrics receive different classification by the experts. This means that those metrics have not sufficient discriminatory power, therefore their weight will be automatically minimized by the learning process. In this sense, rule induction algorithms are particularly robust.

One of the limitations of our method is that it is focused only on individual properties (correctness). If the experts had classified the requirements also according to global properties (completeness, consistency), then the learned rules could not adequately emulate their classification, which would be reflected in a lower effectiveness, as measured by the learning process itself. Extending the method to consider also global properties of requirements is a promising research field that we intend to undertake in future works.

The use of an algorithm to suggest improvements has two obvious risks. First, a change in the requirement text that improves the desired metric can have a bad influence in a different metric, so that the result is still not good, or even worse than before, leading to a new improvement cycle that perhaps could be avoided; this encourages us to find a better algorithm that accounts for both the present requirement and its modified version.

The second risk has a more psychological flavor, it is a phenomenon we can call ‘the bureaucrat engineer’ [Génova & González 2016], an inherent risk of every automatic quality control mechanism. A bureaucrat engineer is one that is satisfied with following the rules, instead of searching for genuine quality. It is the reverse side of a policeman mechanism of penalties in quality management, as opposed to the counselor’s view [James 1999, Génova et al. 2013]. Given a suggestion to improve a requirement, it is probably easy to add or remove some words without scruples such that the modified requirement complies with the rule, even if it becomes nonsense. Nonetheless, we think this does not deprive our proposed method of utility, when it is used as a recommendation system by responsible engineers. We want to emphasize that the method provides not only a single least-effort modification, but a list of suggestions from which the user can follow one that entails a real improvement.

11. Generalizability of the method

In this section we discuss the *generalizability* of our approach, along the lines presented in [Wieringa & Daneva 2015]. These authors emphasize the value of *middle-range theories*, i.e. theories whose generalizations, in contrast to the theories of basic science, do not have universal scope, but which nevertheless give sufficient understanding of a sufficiently large class of cases. The concept of middle-range theory was originally developed for the social sciences [Merton 1968], but is also very well applicable to the engineering sciences [Wieringa 2014].

According to their categorization of strategies to generalize software engineering theories [Wieringa & Daneva 2015], our approach falls in the category of *lab-to-field* strategies, i.e. artifacts that are first developed under idealized laboratory conditions and are then scaled up to operate under uncontrolled field conditions. Besides, our research presents aspects of both sample-based and case-based generalization strategies, as we show below.

The machine learning procedure of finding patterns of quality metrics in a sample of requirements, and using them to predict quality in new samples, is clearly a kind of *statistical learning*, which is one of the sub-categories of *sample-based generalization* strategies identified in [Wieringa & Daneva 2015]. On the other hand, we have used a sample of requirements that originate in a single, even if very reliable, source (experts of the INCOSE's Requirements Working Group); we argue that the good results obtained (the ability of our tool to emulate the experts' judgment) can be similarly expected if the procedure is applied to another source of requirements; we are then exercising *case-based generalization*, which, as the authors explain, is acceptable when it is shown that there exists a reasonable similarity in the underlying mechanisms (what they call "architectural similarity" [Ghaisas et al. 2013, Wieringa & Daneva 2015]). In this case the underlying mechanism is the relationship between measurable indicators of quality in textual requirements and the perceived level of quality by experts, which we have explained in-depth in previous works [Génova et al. 2013].

We can also point to the following *factors of independence* that contribute to the generalizability of the approach:

- *Language independence*. Even if our experiments have been performed with requirements written in English, the method is essentially language-independent, as long as the tool used to extract quality metrics can account for different languages (in our case, the RQA tool provides support for eight different languages: English, French, German, Japanese, Spanish, Swedish, Italian, and Dutch). It is true that the natural-language processing techniques necessary to compute requirements quality metrics have been developed mainly for English, and to a lesser extent for other common languages. The method we have developed depends on the existence of a tool that computes quality metrics, but the method as such does not depend on any particular language or tool.
- *Domain independence.* Quality metrics are domain independent, safe for the last two items 'domain concepts' and 'domain verbs' (see Appendix I). These two metrics certainly depend on the configuration of the auxiliary RQA tool, which must have been provided a suitable ontology of domain terms. However, as long as the ontology has been properly defined, the machine learning procedure to emulate expert quality assessment is, as such, independent of the concrete domain.
- *Size independence.* According to our experiments, the learning algorithms work well with a training set of at least 500 randomly chosen requirements, i.e. the effectiveness reached (accuracy, precision, recall and F-measure) is very close (less than 1% difference) to the values obtained with the whole data set of 1035 requirements. On the other side, once the classifier has been obtained from similar projects with enough requirements from both classes, the approach is

independent of the size of the new project to be evaluated. We have argued before (see Section 4) that the method will be most beneficial for medium/large projects and organizations; but this does not mean that the method will not work in small projects, only that the benefit-cost ratio will probably be not so attractive.

- *Quality levels independence*. The classifier obtained through machine learning techniques is independent of the number of quality levels present in the training data set: it will yield as many levels as they are initially present. Our classifier is binary (two levels, Good and Bad) because that was what the experts provided.
- *Quality demanded independence.* The method to obtain a classifier is independent of the degree of quality demanded in the organization. In fact, what the method achieves is an automatic emulation of the experts' interpretation of quality according to the concrete needs and demands of quality in their particular context, projects and organizations.

These factors of independence are the ground for the alleged tailorablity of the method to different situations, different domains, different styles to write requirements, and different demands in quality.

12. Conclusions and implications

Even if the usages of machine learning techniques to emulate human judgment have been numerous in very different contexts, there are scarce references in the literature, to our knowledge, that describe their application for quality assessment in the field of requirements engineering; that is why we consider our research manifests a high degree of novelty.

Our main contribution in this work has been the development of a method for the flexible evaluation of requirements in an automatic way. We have built an automatic rule-based classifier that emulates the experts' judgment on the quality of requirements and is made of human-readable rules extracted from the expert's tacit knowledge, which can be reworked and improved. The rule generation algorithms take as input an initial set of requirements classified by the experts according to the needs and peculiarities of each project and organization, as well as the quality metrics of those requirements computed with the RQA tool. The generated rules are then applied to automatically assess the quality of new textual requirements, with a high level of agreement between the experts' classification and the automatic classifier (more than 85% in accuracy, precision, recall and F-measure), which is more than enough to provide useful advice to requirements authors.

The experts do not need to make explicit the quality criteria they apply in their assessment of requirements, and the metrics could be computed with a different tool. Therefore, the method is tailorable to different contexts, different styles to write requirements, and different demands in quality, with a high degree of automation. The method is highly general because its results change according to its inputs. The whole process generates a rule-based classifier that emulates the experts' judgment on the quality of requirements (the experts' implicit quality function). These set of quality rules, generated by machine learning, depend both on the set of initial requirements and on the experts' classification of those requirements in several quality levels. Therefore, a different set of classified requirements (i.e. the training data), will produce a different automatic classifier, better adapted to the particular context reflected in the vocabulary of the requirements, and to the particular demands in quality that can be inferred from the experts' classification. In other words, the method builds a classifier that learns from the information provided by the experts, and adapts itself to best emulate them.

Moreover, we can provide a list of automatic recommendations to modify the requirements that have been classified with the lowest quality level, so that they satisfy the obtained quality rules that emulate the experts' judgment. In this way, we are able to point out not only those requirements that need improvement, but also to suggest concrete modifications to achieve the demanded quality; from this list of suggestions the user can choose one that entails a real improvement and at the same time is relatively easy to achieve.

The method is still experimental and has been developed to form an integral part of the Requirements Quality Suite by the Reuse Company [Reuse Company 2016], together with the Requirements Quality Analyzer (RQA) and the Requirements Authoring Tool (RAT). The suite follows a knowledge-centric approach to the requirements definition process, with the application of ontologies and linguistic tools to the analysis and improvement of the quality of specifications. The Requirements Quality Suite has been successfully employed in industry, in companies such as Alstom, as described in [Gallego 2016] and [Chalé-Góngora et al. 2017]. The suite is having a high impact in the domain of Systems Engineering, as reflected in the series of INCOSE (International Council on Systems Engineering) conferences (see for example [Dick et al. 2017]).

The adoption of the method and tool will have some implications for practitioners. First of all, the alleged improvement in quality and flexibility in its assessment, with the economic benefits associated. But, also, requirements authors will have to adapt their working environment, and themselves, to account for the new writing assistant and take advantage of it. This adaptation positively entails, too, that engineers will learn to write requirements in the way the organization wants while they are doing the actual job (*learning on the job* [Marsick & Volpe 1999]). On the negative side, we have already mentioned a possible drift towards the "bureaucratization" of their work (the uncritical submission to a mechanical quality assessment); however, we think the analogy with today-common general-purpose writing assistants allows us not to be pessimistic in this regard. In the end, the best improvement in writing skills will be obtained with a combination of smart tools together with mentoring of requirements authors in basic techniques [Terzakis & Gregory 2016, Gregory & Terzakis 2017].

From a more theoretical viewpoint, this research provides empirical evidence that a non-linear combination of metrics, i.e. the function made of rules obtained through machine learning, provides a better fit to expert judgment than the traditional weighted average of metrics; and it provides empirical evidence, too, that subjective quality (evaluated by experts) can be emulated based on the measurement of objective, low-level features. It is not so obvious that this could be so; a possible "architectural explanation" [Wieringa & Daneva 2015] is that the writer who cares about low level details also cares about high level features of requirements; but this is not more than a conjecture that may well deserve more research.

References

[Aceituna et al. 2014] Aceituna, D., Walia, G., Do, H., Lee S.W. (2014). *Model-based requirements verification method: Conclusions from two controlled experiments*. Information and Software Technology 56(3):321–334.

[Alexander & Stevens 2002] Alexander, I., Stevens, R. (2002). *Writing Better Requirements*. London: Addison-Wesley.

[Ali et al. 2013] Ali, R., Dalpiaz, F., Giorgini, P. (2013). *Reasoning with contextual requirements: Detecting inconsistency and conflicts*. Information and Software Technology 55(1):35–57.

[Belsis et al. 2014] Belsis, P., Koutoumanos, A., Sgouropoulou, C. (2014). *PBURC: A patterns-based, unsupervised requirements clustering framework for distributed agile software development*. Requirements Engineering 19(2):213-225.

[Berry et al. 2006] Berry, D.M., Bucchiarone, A., Gnesi, S., Lami, G., Trentanni, G. (2006). *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.

[Bishop 2006] Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

[Bøegh 2008] Bøegh, J. (2008). *A new standard for quality requirements*. IEEE Software 25(2):57–63.

[Breiman 1996] Breiman, L. (1996). *Bagging predictors*. Machine Learning 24(2):123–140.

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

[Bucchiarone et al. 2005] Bucchiarone, A., Gnesi, S., Pierini, P. (2005). *Quality Analysis of NL Requirements: An Industrial Case Study*. Proceedings of the 13th IEEE International Requirements Engineering Conference, pp. 390-394. Paris, France, August 29 – September 2, 2005.

[Cendrowska 1987] Cendrowska, J. (1987). *PRISM: An algorithm for inducing modular rules*. International Journal of Man-Machine Studies 7(4):349–370.

[Chalé-Góngora et al. 2017] Chalé-Góngora, H.G., Llorens, J., Gallego, E. (2017). *Your Wish, My Command – Speeding up Projects in the Transportation Industry Using Ontologies.* Proceedings of the 27th Annual INCOSE International Symposium (IS 2017), pp. 1070–1086. Adelaide, Australia, July 15-20, 2017.

[Chantree 2006] Chantree, F.J. (2006). *Identifying Nocuous Ambiguity in Natural Language Requirements*. PhD Dissertation, The Open University, Faculty of Maths and Computing, UK.

[Clark & Niblett 1989] Clark, P. and Niblett, T. (1989). *The CN2 induction algorithm*. Machine Learning 3(4):261–283.

[Cleland-Huang et al. 2006] Cleland-Huang, J., Settimi, R., Zou, X., Solc, P. (2006). *The Detection and Classification of Non-Functional Requirements with Application to Early Aspects*. Proceedings of 14th IEEE International Requirements Engineering Conference (RE 2006), Minneapolis, MN, USA, September 11-15 2006, pp. 36–45.

[Cleland-Huang et al. 2007] Cleland-Huang, J., Settimi, R., Zou, X., Solc, P. (2007). *Automated classification of non-functional requirements*, Requirements Engineering 12(2): 103–120.

[De Sousa et al. 2010] De Sousa, T.C., Almeida, J.R., Viana, S., Pavón, J. (2010). *Automatic analysis of requirements consistency with the B method.* ACM SIGSOFT Software Engineering Notes 35(2):1-4.

[Dick et al. 2017] Dick, J., Wheatcraft, L., Long, D., Ryan, M., Llorens, J., Zinni, R., Svensson, C. (2017). *Integrating Requirement Expressions with System Models.* International Council on Systems Engineering (INCOSE), Annual Systems Engineering Conference, ASEC 2017. University of Warwick, Coventry, UK., 21-22 November 2017.

[Dietterich 1997] Dietterich, T.G. (1997). *Machine learning research: four current directions.* Artificial Intelligence Magazine 18(4):97–136.

[Dietterich 2000] Dietterich, T.G. (2000). *An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization.* Machine Learning 40(2):139–157.

[Dollmann & Geierhos 2016] Dollmann, M., Geierhos, M. (2016). *On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements*. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (Austin, Texas, November 1-5 2016), pp. 1807–1816.

[Eito-Brun & Amescua 2017] Eito-Brun, R., Amescua, A. (2017) *Dealing with software process requirements complexity: an information access proposal based on semantic technologies*. Requirements Engineering 22(4): 527–542.

[ESA 1995] European Space Agency. (1995). ESA PSS-05-03. *Guide to the Software Requirements Definition Phase*. ESA Board for Software Standardisation and Control (BSSC) (ftp://ftp.estec.esa.nl/pub/wm/anonymous/wme/bssc/PSS0503.pdf).

[Fabbrini et al. 2001] Fabbrini, F., Fusani, M., Gnesi, S., Lami, G. (2001). *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.

[Fanmuy et al. 2011] Fanmuy, G., Fraga, A., Llorens, J. (2011). *Requirements Verification in the Industry*. Proceedings of the Second International Conference on Complex Systems Design & Management CSDM 2011 (Paris, France, December 7–9, 2011), pp. 145–160.

[Frank & Witten 1998] Frank, E., Witten, I. H. (1998). *Generating accurate rule sets without global optimization.* Proceedings of the 15th International Conference on Machine Learning ICML-98 (Madison, WI, USA, July 24–27, 1998), pp. 144–151.

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

[Gallego et al. 2016] Gallego, E., Chalé-Góngora, H.G., Llorens, J., Fuentes, J., Álvarez, J., Génova, G., Fraga, A. (2016). *Requirements Quality Analysis: A successful case study in the industry.* Proceedings of the Seventh International Conference on Complex Systems Design & Management, CSDM 2016, pp. 187-201. Paris, December 13-14, 2016.

[Génova et al. 2012] Génova, G., Llorens, J., Morato, J. (2012). *Software engineering research: the need to strengthen and broaden the classical scientific method.* In Manuel Mora, Ovsei Gelman, Annette L. Steenkamp and Mahesh Raisinghani (eds.), Research Methodologies, Innovations and Philosophies in Software Systems Engineering and Information Systems. Hershey, PA: IGI Global, 2012, pp. 106-125.

[Génova et al. 2013] Génova, G., Fuentes, J.M., Llorens, J., Hurtado, O., Moreno, V. (2013). *A Framework to Measure and Improve the Quality of Textual Requirements.* Requirements Engineering 18(1):25-41.

[Génova & González 2016] Génova, G., González, M.R. (2016). *Educational Encounters of the Third Kind*. Science and Engineering Ethics 23(6):1791-1800, December 2017.

[Ghaisas et al. 2013] Ghaisas, S., Rose, P., Daneva, M., Sikkel, K., Wieringa, R. (2013). *Generalizing by similarity: lessons learnt from industrial case studies*. Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry, CESI'13. Held at ICSE'13, 35th International Conference on Software Engineering, San Francisco, CA, USA, May 18-26, 2013, pp. 37-42.

[Gregory & Terzakis 2017] Gregory, S., Terzakis, J. (2017). *Viewpoint: effectiveness of focused mentoring to improve requirements engineering industrial practice.* Requirements Engineering 22(3): 413–417.

[Harman & Jones 2001] Harman, M., Jones, B.F. (2001). *Search-based software engineering*. Information and Software Technology 43(14):833–839.

[Harman et al 2012] Harman, M., McMinn, P., De Souza, J., Yoo, S. (2012). *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*. Empirical Software Engineering and Verification. International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures. Springer Lecture Notes in Computer Science, 7007, pp. 1–59.

[Heck & Zaidman 2018] Heck, P., Zaidman, A. (2018). *A systematic literature review on quality criteria for agile requirements specifications*. Software Quality Journal 26(1):127–160.

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

[Hong et al. 1986] Hong, J., Mozetic, I. and Michalski, R.S. (1986). *AQ15: Incremental learning of attribute-based descriptions from examples, the method and user's guide*. Report ISG 85-5, UIUCDCS-F-86-949, Department of Computer Science, University of Illinois at Urbana-Champaign.

[Hooks 1993] Hooks, I. (1993). *Writing good requirements*. Proceedings of the 3rd NCOSE International Symposium, vol. 2, pp. 1–12.

[Huang et al. 2015] Huang, L., Ng, V., Persing, I., Chen, M., Li, Z., Geng, R., Tian, J. (2015). *AutoODC: Automated generation of orthogonal defect classifications*. Automated Software Engineering 22(1): 3–46.

[Hussain et al. 2007] Hussain, I., Ormandjieva, O., Kosseim, L. (2007). *Automatic quality assessment of SRS text by means of a decision-tree-based text classifier*. Proceedings of the Seventh International Conference on Quality Software (QSIC 2007), Portland, OR, 11-12 Oct. 2007, pp. 209–218.

[Hussain et al. 2008] Hussain, I., Kosseim, L., Ormandjieva, O. (2008). *Using linguistic knowledge to classify non-functional requirements in SRS documents*. Proceedings of the International Conference on Application of Natural Language to Information Systems, NLDB 2008, London, UK, June 24-27 2008. Springer Lecture Notes in Computer Science, 5039, pp. 287–298.

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

[IEEE Computer Society 2014] IEEE Computer Society. (2014). *SWEBOK Guide to the Software Engineering Body of Knowledge*, Version 3.0. (http://www.computer.org/portal/web/swebok).

[INCOSE 2012] International Council on Systems Engineering (INCOSE), Requirements Working Group. (2012). *Guide for Writing Requirements*. (http://www.incose.org/ProductsPublications/techpublications/GuideRequirements).

[ISO/IEC 2007] ISO/IEC 25030:2007. *Software engineering — Software Product Quality Requirements and Evaluation (SQuaRE) — Quality requirements* (http://www.iso.org/iso/catalogue_detail.htm?csnumber=35755).

[James 1999] James, L. (1999). *Providing Pragmatic Advice On How Good Your Requirements Are - The Precept 'Requirements Councillor' Utility*. Proceedings of the 9th INCOSE International Symposium, Brighton, England, 6–11 June 1999, pp. 1427–1430.

[Jani & Islam 2012] Jani, H., Islam, A. (2012). *A Framework of Software Requirements Quality Analysis System using Case-Based Reasoning and Neural Network.* (2012). Proceedings of the 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM), Taipei, 23-25 Oct. 2012, pp. 152-157.

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

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

[Ko et al. 2007] Ko, Y., Park, S., Seo, J., Choi, S. (2007). *Using classification techniques for informal requirements in the requirements analysis-supporting system*. Information and Software Technology 49(11-12):1128–1140.

[Kohavi 1995] Kohavi, R (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence IJCAI-95 (Montreal, Quebec, Canada, August 20–25, 1995), vol. 2, pp. 1137–1143.

[Loucopoulos & Karakostas 1985] Loucopoulos, P., Karakostas, V. (1985). *Systems Requirements Engineering*. New York: McGraw-Hill.

[Lucassen et al. 2016] Lucassen, G., Dalpiaz, F., Van Der Werf, J.M., Brinkkemper, S. (2016). *Improving agile requirements: the Quality User Story framework and tool.* Requirements Engineering 21(3):383-403.

[Magee & Tripp 1997] Magee, S., Tripp, L.L. (1997). *Guide to software engineering standards and specifications*. Boston: Artech House.

[Major & Mangano 1995] Major, J.A. and Mangano, J.J. (1995). *Selecting among rules induced from a hurricane database*. Journal of Intelligent Information Systems 4(1):39–52.

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

[Merton 1968] Merton, R. (1968). *On sociological theories of the middle range*. In: Social Theory and Social Structure, enlarged edition. New York: The Free Press, 1968, pp. 39–72.

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

[Moreno et al. 2016] Moreno, V., Génova, G., Parra, E., Fraga, A. (2016). *Metrics obtained with the RQA tool from a set of 1035 requirements provided by INCOSE Requirements Working Group*. July 2016 (available at https://gonzalogenova.files.wordpress.com/2020/02/incose-rqa-20-metrics-1035-requirements.pdf).

[Otero et al. 2010] Otero, C.E., Dell, E., Qureshi, A., Otero, L.D. (2010). *A quality-based requirement prioritization framework using binary inputs*. Proceedings of the 4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, Kota Kinabalu, Malaysia, 26-28 May 2010, pp. 187–192.

[Ott 2013] Ott, D. (2013). *Automatic requirement categorization of large natural language specifications at Mercedes-Benz for review improvements*. Requirements Engineering: Foundation for Software Quality. Proceedings of the 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Springer Lecture Notes in Computer Science, 7830, pp. 50–64.

[Parra et al. 2015] Parra, E., Dimou, C., Llorens, J., Moreno, V. and Fraga, A. (2015). *A methodology for the classification of quality of requirements using machine learning techniques.* Information and Software Technology 67:180–195.

[Popescu et al. 2008] Popescu, D., Rugaber, S., Medvidovic, N., Berry, D.M. (2008). *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 Lecture Notes in Computer Science, 5320, pp. 103-124.

[Quinlan 1986] Quinlan, J.R. (1986). *Induction of Decision Trees (ID3 algorithm).* Machine Learning 1(1):81–106.

[Quinlan 1993] Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.

[Rashwan 2015] Rashwan, A. (2015). *Automated Quality Assurance of Non-Functional Requirements for Testability*. Masters thesis, Concordia University, Montréal, Québec, Canada, April 2015.

[Reuse Company 2016] The Reuse Company. (2016). *RQA Requirements Quality Analyzer* (http://www.reusecompany.com/requirements-quality-analyzer).

[Robledano et al. 2016] Robledano, J., Moreno, V., Pereira, J.M. (2016). *Aproximación experimental al uso de métricas objetivas para la estimación de calidad cromática en la digitalización de patrimonio documental gráfico*. Revista Española de Documentación Científica 39(2):e128.

[Rosenberg & Linda 2001] Rosenberg, L.H., Linda, H. (2001). *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.

[Russell & Norvig 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.

[Sardinha et al. 2013] Sardinha, A., Chitchyan, R., Weston, N., Greenwood, P., Rashid, A. (2013). *EA-Analyzer: Automating conflict detection in a large set of textual aspect-oriented requirements*. Automated Software Engineering 20(1):111–135.

[Schapire 1990] Schapire, R.E. (1990). *The strength of weak learnability*. Machine Learning 5(2):197–227.

[Siahaan & Umami 2011] Siahaan, D., Umami, I. (2011). *Natural Language Processing for Detecting Forward Reference in a Document*. IPTEK, The Journal for Technology and Science 22(4):138–142.

[Terzakis & Gregory 2016] Terzakis, J., Gregory, S. (2016). *RAMP: Requirements Authors Mentoring Program.* Proceedings of the 24th International Requirements Engineering Conference (RE 2016), Beijing, China, September 12-16 2016, pp. 323–328.

[Thakurta 2013] Thakurta, R. (2013). *A framework for prioritization of quality requirements for inclusion in a software project*. Software Quality Journal 21(4):573–597.

[The Standish Group 2015] The Standish Group. (2015). *Chaos Report* (http://www.standishgroup.com/).

[Thitisathienkul & Prompoon 2015] Thitisathienkul, P., Prompoon, N. (2015). *Quality Assessment Method for Software Requirements Specifications Based on Document Characteristics and Its Structure*. Proceedings of the Second International Conference on Trustworthy Systems and Their Applications (TSA 2015), Hualien, Taiwan, July 8-9 2015, pp. 51–60.

[Turk 2006] Turk, W. (2006). *Writing requirements for engineers*. IET Engineering Management 16(3):20–23.

[Wang et al. 2013] Wang, Y., Gutiérrez, I.L.M., Winbladh, K., Fang, H. (2013). *Automatic Detection of Ambiguous Terminology for Software Requirements*. In Natural Language Processing and Information Systems, Proceedings of the 18th International Conference on Applications of Natural Language to Information Systems, NLDB 2013, Salford, UK, June 19-21, 2013. Springer Lecture Notes in Computer Science, 7934, pp. 25-37.

[Weiss & Indurkhya 1998] Weiss, S.M., Indurkhya, N. (1998). *Predictive data mining: a practical guide*. San Francisco: Morgan Kaufmann.

[Wieringa 2014] Wieringa, R. (2014). *Design Science Methodology for Information Systems and Software Engineering*. New York: Springer.

[Wieringa & Daneva 2015] Wieringa, R., Daneva, M. (2015). *Six strategies for generalizing software engineering theories*. Science of Computer Programming 101:136-152.

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

[Witten & Frank 2000] Witten, I.H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann.

[Wolpert 1992] Wolpert, D.H. (1992). *Stacked generalization.* Neural Networks 5(2):241–259.

[Zhang et al. 2007] Zhang, Y., Harman, M., Mansouri, S. (2007). *The Multi-Objective Next Release Problem*. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07), London July 7-11, 2007, pp. 1129-1137.

APPENDIX I – Description of quality metrics

For a more complete description and justification of the metrics, see our previous works [Génova et al. 2013, Parra et al. 2015].

Morphological	
Paragraphs	The requirement should not be expressed in too many paragraphs to avoid over-specification, redundancy of information and expression of various needs in the same requirement.
Words	The requirement should not have an excessive number of words to avoid the same problems related to size.
Readability	Readability measures the degree of difficulty to read a text, based on average syllables/characters per word and average words per sentence. Bad readability may cause confusion.
Punctuation	Measured as the number of characters between punctuation marks. Incorrect punctuation hinders the readability of the requirement.
Lexical	
Connectors	Number of copulative-disjunctive connectors in the requirement. Using multiple connectors may indicate different needs in the same requirement and therefore compromise the atomicity.
Negative	Negative expressions can make the requirement hard to understand.
Control-flow	The requirement should avoid pseudocode and control-flow expressions to avoid specifying the solution to the problem.
Implicit	The requirement should be explicit, avoid the use of personal pronouns.
Ambiguous	Using ambiguous expressions may render the requirement difficult to understand.
Incomplete	Sentences such with incomplete enumerations (‘etc.’, ‘not limited to’, ‘as a minimum’) demonstrate the requirement has not a clear scope or it is not atomic.
Speculative	The use of speculative expressions (‘enough’, ‘sufficient’, ‘approximately’, etc.) may indicate that the real need of the requirement is not clear.
Rationale	Avoid justifications in the requirement.
Design	The requirement should focus on a necessity instead of expressing a solution.
Analytical	
Imperative	The requirement should have at least one imperative verb.
Conditional	The requirement should be written in assertive way.
Passive voice	The use of verbs in passive voice may hinder the understanding of the requirement.
Domain concepts	A large number of domain concepts used in the same requirement can indicate over-specification.
Domain verbs	Too many domain verbs may indicate that the requirement has expressed too many needs.