Toward a Text Classification System for the Quality Assessment of Software Requirements Written in Natural Language

Olga Ormandjieva
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
ormandj@cse.concordia.ca

Leila Kosseim
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
kosseim@cse.concordia.ca

Ishrar Hussain
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
h_hussa@cse.concordia.ca

ABSTRACT
Requirements Engineering (RE) is concerned with the gathering, analyzing, specifying and validating of user requirements that are documented mostly in natural language. The artifact produced by the RE process is the software requirements specification (SRS) document. The success of a software project largely depends on the quality of SRS documentation, which serves as an input to the design, coding and testing phases. This paper approaches the problem of the automatic quality assessment of textual requirements from an innovative point of view, namely the use of the Natural Language Processing (NLP) text classification technique. The paper proposes a quality model for the requirements text and a text classification system to automate the quality assessment process. A large study evaluating the discriminatory power of the quality characteristics and the feasibility of an annotated tool for the automatic detection of ambiguities in requirements documentation is presented. The study also provides a benchmark for such an evaluation and an upper bound on what we can expect automatic requirements quality assessment tools to achieve. The reported research is part of a larger project on the applicability of NLP techniques to assess the quality of artifacts produced in RE.

Categories and Subject Descriptors
D.2.9 [**Management**]: Software quality assurance (SQA)

General Terms
Management, Reliability, Experimentation.

Keywords
Requirements Engineering, Quality Assessment, Natural Language Processing, Human Annotation, Text Classification Techniques.

1. INTRODUCTION
Software requirements specification (SRS) documents are the medium used to communicate user requirements to the technical people responsible for developing the software. Requirements analysis and validation constitute the key requirements engineering (RE) activity for understanding the user requirements gathered, for classifying them and for relating stakeholders' needs to possible software requirements. This process often takes a considerable time to perform manually, as the length of a real-life requirements document can range from a few pages to hundreds of pages containing numerous words, phrases and sentences, where each can potentially be wrongly interpreted. Consequently, checking for errors manually, although the most common way of doing so, is also one of the costliest phases of RE. For all these reasons, Natural Language Processing (NLP) techniques have been developed to tackle this problem.

This paper addresses a problem related to the automatic quality assessment of textual requirements documents. Such (semi-) automatic assessment can reduce the time needed for requirements analysis and validation in the requirements specification phase, and ultimately increase the quality of the SRS documentation. This should help software engineers correctly understand the problem and develop the right solution for it.

The work presented in the paper is part of a bigger project aimed at applying NLP techniques to the RE process (see Figure 1). The objective of NLP assessment in the context of that project can be expressed in terms of three main goals:

G1. Automatic NLP-driven quality assessment of the textual requirements in the requirements gathering and elicitation phase.

G2. Automatic NLP-driven quality assessment of the textual requirements in the analysis and specification phase, where conceptual static and dynamic models are developed from the textual requirements.

G3. Graphical visualization and animation of the conceptual models extracted from the requirements text for the user's validation and feedback.

Research hypothesis. The research described in this paper is concerned with the challenges inherent in understanding the

initial textual requirements (see G1 above) using the NLP text classification technique. The objective is to identify the textual ambiguities in the requirements elicitation phase before the conceptual modeling of the requirements begins. Our hypothesis is that the root cause of errors being introduced into the requirements (and the consequent reduction in quality) is ambiguity in the text. Here, we define ambiguity as the difference between the depiction of an informal textual description of the problem (requirement) and the description of the solution for the informal domain where the intents lie. We affirm that lowering the level of ambiguity in the textual requirements document will lead to a better quality conceptual description (model) of the solution, and also reduce the amount of time required for requirements analysis and specification.

Approach. We have developed a quality model for requirements quality assessment derived from the existing guidelines in the literature for writing SRS documentation (such as [6,8,10,15,18,19]) and from the authors' experience. The quality of the requirements text is analyzed from two different points of view, namely surface (literal) understanding and conceptual (modeling) understanding. The objective is to apply the NLP text classification technique to build a system for the automatic detection of ambiguity in requirements documents based on the quality indicators defined in the quality model. We believe that, with proper training, such a text classification system will prove to be of immense benefit in detecting ambiguities in a software requirements text.

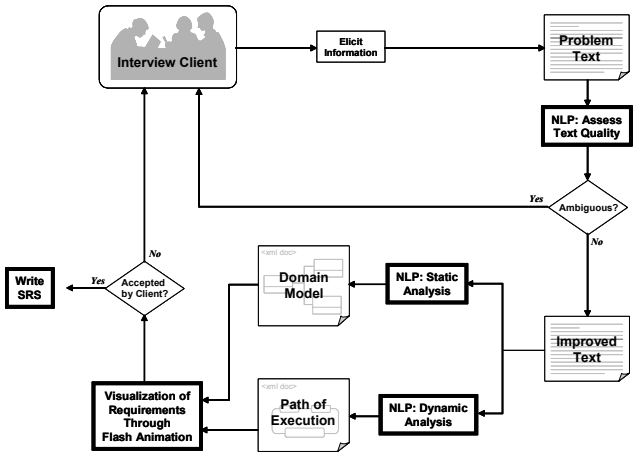

Figure 1. NLP-Based Quality Assessment in the Requirements Engineering.

For this task, a body of requirements documents was built up and annotated for ambiguity from the point of view of both a surface and a conceptual understanding. We present a study in which the feasibility of (semi)-automatic tools is evaluated by assessing how difficult the task of ambiguity assessment of textual requirements really is, and how the use of automatic tools compares to human performance. Experiments have been performed and quality data have been statistically analyzed to demonstrate the feasibility of using the quality model, and to provide quantitative bases for interpreting the surface understanding data and deciding whether or not a requirements document is of acceptable quality. To the authors' knowledge, this is the first attempt in the literature to apply the NLP text classification technique to software requirements quality assessment.

The paper is organized as follows: Section 2 introduces the proposed quality model for SRS textual documentation. The experimental study on quality assessment and the discussion of the research results are outlined in section 3. A critique of our research results in comparison to related work is given in section 4. Finally, our conclusions and directions for future work are outlined in section 5.

2. QUALITY MODEL

Stakeholders involved in the use and development of a system should be able to understand the requirements text at both the surface and conceptual levels. Writing requirements that are unambiguous at both levels is critical in the software life cycle. If not detected early, ambiguities can lead to misinterpretations at the time of requirements analysis and specification, or at a later phase of the software development life cycle, causing an escalation in the cost of requirements elicitation and software development. Detecting ambiguities at an early stage of the requirements elicitation process can therefore save a great deal aggravation, not to mention cost.

Comprehension of the requirements text describing a problem and its domain can typically be divided into two broad levels: the literal meaning (or surface understanding) and the interpretation (or conceptual understanding). In the context of our work, we consider surface understanding and conceptual understanding to be the two main factors on which the quality of a text depends. The decomposition of the above two factors into the corresponding quality criteria is shown in Figure 2.

We use the term "surface understanding" to represent how easy or how difficult it is to understand the facts stated in the document, without judging its design or implementation concerns in terms of any software engineering concept. Reading at this level means understanding the facts stated in the document. It allows us to answer basic questions such as who, what, when and where. Several surface factors can be involved at this level of understanding; e.g. sentence length, ambiguous adjectives and adverbs, passive verbs, etc., and all the features necessary for a surface understanding can be categorized into two major sets, based on their scope of effectiveness: (1) Sentence-level features; and (2) Discourse-level features.

By contrast, we use the term "conceptual understanding" to represent how much a developer would gain in designing or implementing a system by carefully reading/examining its problem texts only. The conceptual level involves interpretation of the document: understanding what is meant or implied, rather than what is stated. This includes making logical links between facts or events, drawing inferences and trying to represent the content more formally. This level of understanding involves deeper factors, such as the "seven sins of the specifier" described by Meyer [15]. His comprehensive study presents a thorough description of such mistakes by classifying them into seven distinct categories or "sins", as he calls them. These sins are reproduced in Table 1.

Table 1. Meyer's seven sins of the specifier [15].

Noise	The presence in the text of an element that does not carry information relevant to any feature of the problem.
Silence	The existence of a feature of the problem that is not covered by any element of the text.
Over-specification	The presence in the text of an element that does not correspond to a feature of the problem, but to a feature of a possible solution.
Contradiction	The presence in the text of two or more elements that define a feature of the system in an incompatible way.
Ambiguity[1]	The presence in the text of an element that makes it possible to interpret a feature of the problem in at least two different ways.
Forward Reference	The presence in the text of an element that uses features of the problem not defined until later in the text.
Wishful Thinking	The presence in the text of an element that defines a feature of the problem in such a way that a candidate solution cannot realistically be validated with respect to this feature.

Ideally, the linguistic quality of a requirements document should be assessed automatically, or at least helped by automation. In fact, a great deal of research has been conducted with that goal in mind (see section 4). However, before automating the process, it was important to study our quality model and determine whether or not automation would be feasible. The details of that study are presented in section 3.

3. QUALITY ASSESSMENT FEASIBILITY: A STUDY

To evaluate the feasibility of our quality assessment approach, we asked four human annotators to manually classify requirements documents based on the text quality indicators (see Figure 2), and then we measured the extent to which they agreed or disagreed with our approach. These indicators were selected consulting similar studies of [5,6,8,10,15,18,19]. Our premise was that, if humans agree statistically on the quality of requirements texts, then the quality model truly measures what it is supposed to measure; namely, the quality of the textual description of the requirements. By contrast, if the human annotators cannot statistically agree on a classification, then the automation would be difficult to achieve and it would not be possible to evaluate the results of the automatic classification. The details of the study are given below.

3.1 Design of the experiment

To perform the manual classification, we asked four annotators to read and categorize a set of requirements documents. All the annotators had a software engineering background, but in different fields of computer science. We gave them specific guidelines based on our quality model and clear examples of what was to be considered ambiguous, looking at surface understanding and conceptual understanding separately. The annotators were to score all the passages of our documentation (on a scale from 0 to 10, the higher the score, the less ambiguous the passage). The annotation guidelines indicated what to look for in a passage, but did not give any strict instructions on the scoring, in order to give the annotators the freedom to score as they saw fit. The annotation task took about 7 hours of effort per person for both surface understanding and conceptual understanding. On average, it took 2.5 minutes to rank each passage. Considering that each passage contains, on average, 189 words, the task was a time-consuming one.

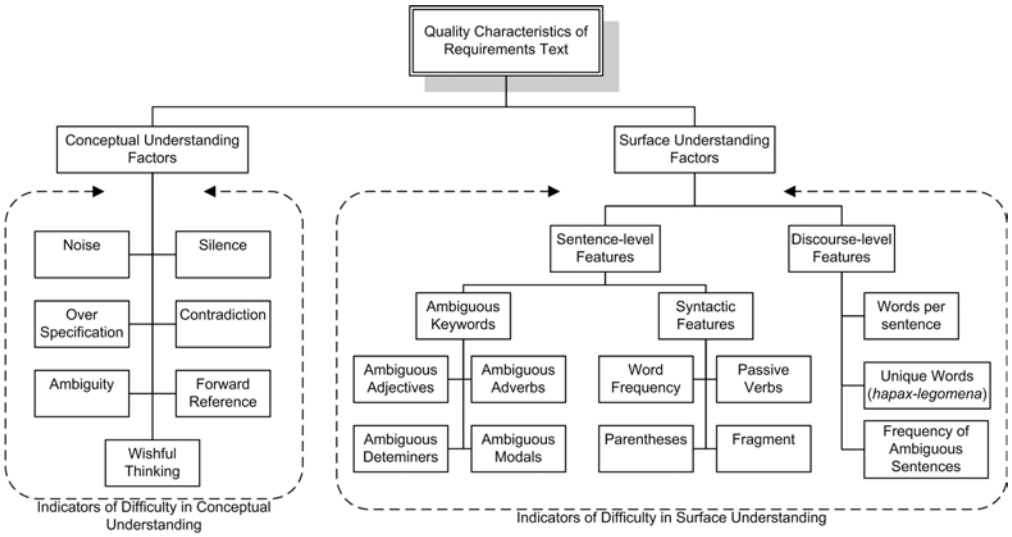

Figure 2. Quality model for software requirements text.

[1] Meyer's use of the term *ambiguity* in [15] is different from the way *ambiguity* is used in our work. In our case, we use it in a broader sense.

3.2 Discussion of the results

To analyze the results, we first translated all the scores of the annotators into a binary decision: "U" (unambiguous) or "A" (ambiguous), according to the standard interpretation. The translated grades of "U" and "A" were then used to compute the gold standard as the majority vote for each passage after removing undecided votes. The results of the gold standard computation are shown in Table 1. Of the 165 passages, 153 (92.7%) were classified as "U" (unambiguous) for surface understanding, 12 (7.3%) as "A" (ambiguous) and there were no undecided votes. For conceptual understanding, 138 (83.6%) of the 165 samples were unambiguous, 27 (16.4%) were ambiguous and there were no undecided votes. As one would expect, conceptual understanding seems harder to achieve than surface understanding.

Table 2. The gold standard.

	Unambiguous (U)	Ambiguous (A)	Undecided
Surface Understanding	92.7%	7.3%	0%
Conceptual Understanding	83.6%	16.4%	0%

The purpose of the annotation was to find out whether or not humans are able to agree on a classification scheme to determine if the task is amenable to automation. To compute inter-annotator agreement, we used the Kappa index, introduced by Cohen [3]. Figure 3 shows the results of this analysis, revealing how strongly each annotator agrees with the majority decision, i.e. the gold standard. Here, the average Kappa simply reflects the strength of the gold standard itself.

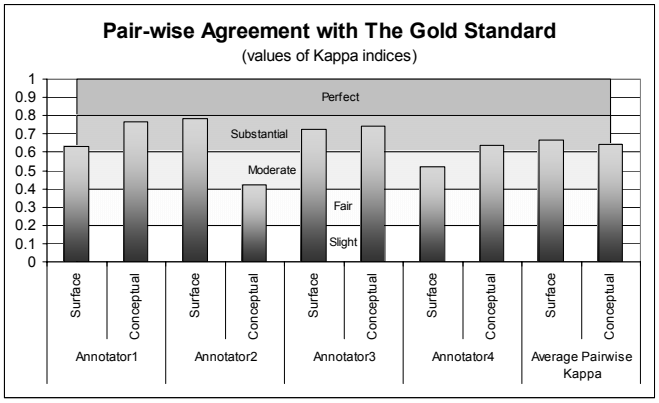

Figure 3. Pairwise inter-annotator agreement with the gold standard.

According to the interpretation of Kappa values given by Landis *et al*. [11], on average, the annotators tend to agree with the gold standard to a "Substantial" degree on both the surface and conceptual levels. For annotators 2 and 4, however, the Kappa values indicate a "Moderate" level of agreement. Overall, the Kappa statistic shows that annotators tend to agree with one another, which proves the feasibility of the quality assessment of requirements text proposed in this paper.

The results are sufficient for us to believe that an automatic system can be built to emulate the decision-making process of the human annotators and to automatically classify requirements documents. However, a high level of precision should not be expected in this task. The average inter-annotator agreement indicated by the kappa values of 0.66 for surface understanding and 0.64 for conceptual understanding should be seen as upper bounds on the accuracy of any classifier.

In addition, the analysis indicates a positive correlation between the surface and conceptual understanding of the text, and a negative correlation between the understanding and the time required to analyze a text. The above confirms our hypothesis that lowering the level of surface ambiguity would lead to a better conceptual understanding of the requirements and reduce the time needed for requirements analysis. This emphasizes the importance of our research results in the RE field.

3.3 Toward a text classification system

Our current research objective, as a "proof of concept" and thus part of our feasibility study, was to build a text classification system that could classify sentences as "ambiguous" or "unambiguous", in terms of surface understanding. Although our ultimate target was to build a classifier that can classify a discourse in terms of its ambiguity, we focused on building a similar classifier at sentence level to assess the achievability of automating the task of ambiguity detection at the more limited scope of the sentence.

We have succeeded in extracting the values of almost all possible quality indicators from all our samples. We developed a sentence-level Feature Extractor tool written in Java which extracts the values of features (indicators) likely to make a sentence "ambiguous" or "unambiguous", in terms of surface understanding (see "Indicators of Difficulty in Surface Understanding"). The Feature Extractor tool then feeds the sentences one-by-one to the Stanford Parser [9] for POS tagging and syntax parsing. The values of the indicators mentioned above are then counted for each sentence. We chose the C4.5 decision tree learning algorithm for the classification task. The two main reasons for this choice were: (1) Decision trees can allow backtracking from a leaf to derive the cause of a particular classification, and C4.5 (revision 8), with its post-pruning feature, was the best open-source decision tree learning algorithm available to us; (2) The size of the documentation was not large enough for training neural network algorithms, which would have yielded better results.

We have determined the discriminating power of the surface understanding indicators, and have developed a classifier to actually flag ambiguous and unambiguous texts at the surface level (see Figures 4, 5).

It should be noted that the tree in Figure 4 was dynamically generated, which means that, with the introduction of new training data, the classifier is able to generate new decision trees. The accuracy of our sentence classifier establishes its applicability in practical fields, where ambiguity is detected at sentence level.

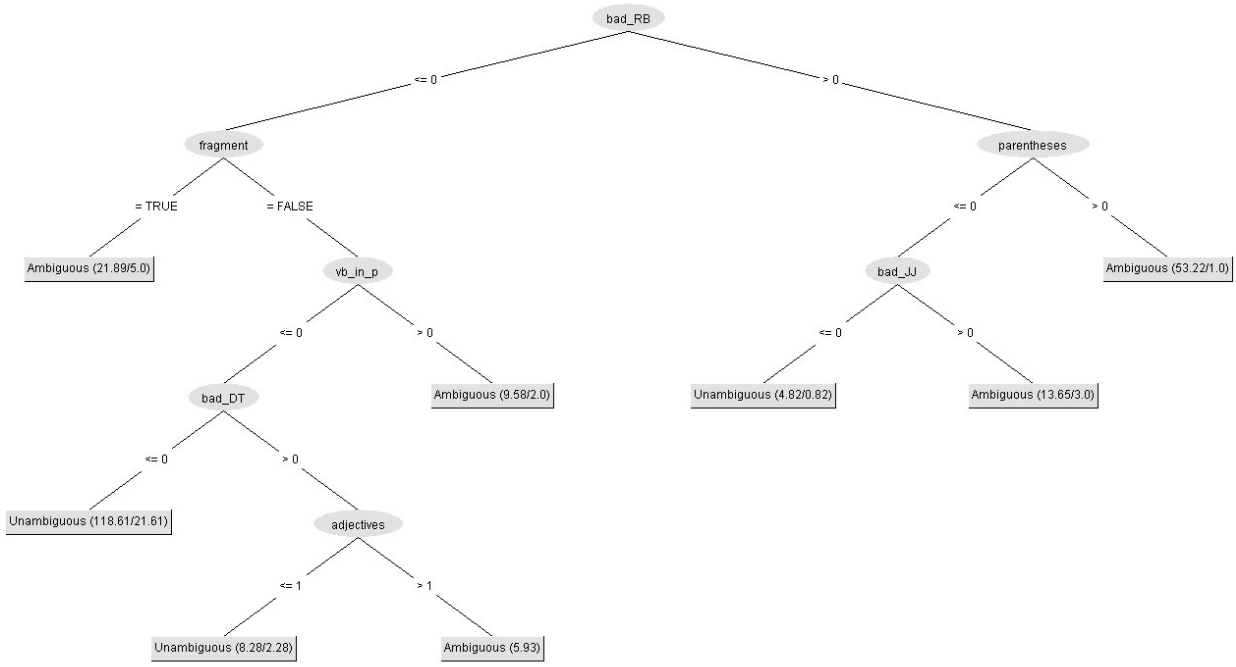

Figure 4. Decision Tree generated by the C4.5 algorithm for Sentence Classification

Again, we chose the C4.5 (revision 8) decision-tree learning algorithm for building the discourse-level classifier. The classifier requires that three feature values be extracted from a passage of unknown status, so that it can predict its classification as the nominal values: "ambiguous" or "unambiguous". The decision tree generated by the aforementioned C4.5 algorithm after training is shown in the following figure:

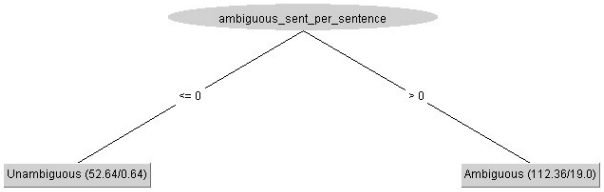

Figure 5. Decision tree generated by the C4.5 learning algorithm after training with discourse-level feature

The tree contains the single feature, "ambiguous_sent_per_sentence" (i.e. the density of ambiguous sentences, or the number of ambiguous sentences, divided by the total number of sentences in a section) at its root. No other feature is included in the tree by the C4.5 algorithm, affirming that this feature alone is sufficient for the classification task. We embedded the C4.5 decision-tree learner with our discourse classifier to keep the learning process dynamic, and to check the applicability of the other, comparatively weaker, classification features every time, when learning from new training data.

On this initial experiment, the discourse classifier resulted in an accuracy of 86.67% agreeing with the human annotations (when trained and tested with 10-fold-cross-validation method). These results affirm that it is indeed possible to detect ambiguity in terms of surface understanding by means of currently available NLP tools and text classification techniques.

4. RELATED WORK

Many studies have previously addressed the issue of detecting ambiguities in requirements documents, and several approaches have been proposed. Although they are often similar in the types of tools they use, these approaches are sometimes radically different in the way they attempt to detect ambiguities.

The use of manual inspection still seems to be the most popular way to detect and resolve ambiguities. A leading study, and one of the earliest, in this field was conducted by Bertrand Meyer [15], who stressed that natural language requirements specifications are inherently ambiguous, and that the use of formal specifications is absolutely necessary to resolve these ambiguities. Meyer's approach to detecting such ambiguities was to inspect each word, phrase and sentence manually. For their part, Kamsties *et al*. [8] proposed a specific methodology of human inspection to resolve ambiguity. While they argue in favor of manual inspections, their work demonstrates a dependence on formal specifications, e.g. UML models, especially for detecting ambiguities related to the problem domain. Their study concludes that "one cannot expect to find all ambiguities in a requirements document with realistic resources" – even with such complete human involvement. Manual detection is typically the most accurate approach; however, it is also the most expensive. We also note that Letier *et al*. [12] propose the use of formal specifications to validate requirements.

The work of Ambriola *et al*. [1] attempts to validate NL Specification with the help of the user after deriving a conceptual model automatically from the requirements specifications using their tool, which they call Circe. This tool is funded by IBM and is now available as a plug-in for Eclipse. Although Circe is in general use, it still does not consider the existence of ambiguities at the level of surface understanding. This could corrupt their

conceptual model, making the errors extremely difficult for a user to detect from the model later on.

Many other studies attempt to reduce the problems associated with unrestricted NL by limiting the scope of the language. Some use a new NL-like sublanguage, as in [4,11,14], but this is not truly NL. Others propose restricting the grammar to consider only a subset of NL when writing a requirements specification [4,5,7,17]. However, although using a restricted language does simplify the task of detecting ambiguities, it imposes severe constraints on the software engineer's freedom of expression.

Recently, researchers have attempted to deal with unrestricted language by using techniques developed in NL processing (NLP). Tools such as part-of-speech taggers, syntactic parsers and named-entity taggers have achieved very respectable accuracies, which means that they can be used for real-world texts. Osborne *et al*. [16], for example, try to detect ambiguities in SRS documents through syntax. They use a syntactic parser to derive all possible sentence parsing trees. If a sentence generates more than one parsing tree, then it is considered ambiguous. The problem with this approach is that what is possible at the syntactical level may not be plausible at the interpretation level. Discourse or world-knowledge constraints may eliminate a possible syntactical interpretation, leaving a sentence with multiple syntactical parses which are unambiguous to the human reader.

Another interesting tool is that of Wilson *et al*. [18,19], which uses nine quality indicators for requirements specification: Imperatives, Continuances, Directives, Options, Weak Phrases, Size, Specification Depth, Readability and Text Structure. However, results derived from using their tool show only the frequency counts of those indicators in different samples, without taking the crucial decision of whether or not a sample is ambiguous.

Fabbrini *et al*. [6,10] address the issue by proposing a tool called "QuARS: Quality Analyzer for Requirements Specification". QuARS syntactically parses the sentences using the MINIPAR parser [13], then it combines both lexical (part-of-speech tags) and syntactical information to detect specific ambiguity indicators of poor-quality requirements specification. In their paper, however, the quality indicators seem to be mostly based on specific keywords, rather than on more general classes of words. At every stage of processing, QuARS requires the use of a different "modifiable" dictionary, which seems to be manually created and modified for a particular stage of processing and for a specific problem domain by the requirements engineer. Their idea seems to be dependent on using these special dictionaries, the relevance and practical usefulness of which are uncertain. Again, their quality measurement metrics are not well enough defined to characterize a text as ambiguous.

As discussed, researchers have previously attempted to flag ambiguous texts using various (semi-) automatic methods. However, these methods have typically been evaluated anecdotally or on a small scale. To our knowledge, no one has attempted a formal evaluation of their results and a comparison to human evaluations. Our study (see section 3) evaluates the feasibility of such a task by analyzing how difficult it really is to perform and how the automatic tools developed can compare to human performance. Our work provides a benchmark for such an evaluation and an upper bound on what we can expect automatic tools to achieve.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a hierarchical quality model for the decomposition of software requirements text quality into measurable features that would be collected directly from the text. The quality model targets the automatic assessment of textual requirements in terms of their ambiguity. A study investigating the feasibility of the quality characteristics and of an annotated tool for automatic detection of ambiguity in requirements documents is also presented. The results show that the quality assessment task is difficult for humans, but there is substantial agreement on the chosen quality indicators, both at the level of surface understanding and at the level of conceptual understanding. We have developed a classifier to actually flag ambiguous and unambiguous texts at the surface level of understanding. The above demonstrates the feasibility of our quality assessment approach.

A thorough analysis of the annotators' data led to the following conclusions:

1) Most of the indicators in our quality model are very good at detecting ambiguity at the level of surface understanding, which proves that they are objective indicators of surface understanding. All these indicators can be extracted automatically from the text using a POS tagger and a parser.

2) We found virtually no conceptual characteristic that would be extractable by currently available NLP tools for discriminating ambiguities of conceptual understanding. This may be due to the subjective nature of the conceptual understanding process, which requires expertise in RE. Therefore, more studies will be required to identify objective indicators for SRS conceptual understanding criteria. This will be addressed in our future work in this direction.

Our future work includes the development of a system for the conceptual understanding of requirements text. We believe that such a classifier would be very beneficial to software RE. The ability to detect serious ambiguities in the requirements text at a very early stage of requirements elicitation could significantly reduce both expense and aggravation, and could help avoid very costly misinterpretations. The tool should not only work in a standalone mode, but should be incorporated into a formal model-building system for NLP-based quality assessment in the RE phase. A thorough evaluation of the accuracy of the text classification system needs to be performed with a large set of requirement documents, and the results compared to human performance. We plan to tackle this in our future work.

6. REFERENCES

[1] Ambriola, V., Gervasi, V., "Processing natural language requirements," In proceedings of Automated Software Engineering (ASE'97): 12th IEEE International Conference, November 1-5, 1997, pp. 36-45, 1997.

[2] Carletta, J., "Assessing agreement on classification tasks: The kappa statistic," Computational Linguistics, 22(2), 1996, pp. 249–254.

[3] Cohen, J., “A coefficient of agreement for nominal scales,” Educational and Psychological Measurement, 20, 1960, pp. 37-46.

[4] Cyre, W. R., “A Requirements Sublanguage for Automated Analysis,” International Journal of Intelligent Systems, 10 (7), pp. 665-689, July 1995.

[5] Denger, C., Berry, D., Kamsties, E., "Higher Quality Requirements Specifications through Natural Language Patterns," SWSTE, p. 80, IEEE International Conference on Software-Science, Technology & Engineering, 2003.

[6] Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G., “An Automatic Quality Evaluation for Natural Language Requirements,” Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'01, Interlaken, Switzerland, June 4-5, 2001.

[7] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., and Moreschini, P., “Assisting requirement formalization by means of natural language translation,” Formal Methods in System Design, vol. 4, pp. 243-263, 1994.

[8] Kamsties, E., Berry, D.M., and Paech, B., “Detecting Ambiguities in Requirements Documents Using Inspections,” p. 68-80 in Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01), Paris, France, July 23, 2001.

[9] Klein, D. and Manning, C. D., “Accurate Unlexicalized Parsing”, Proceedings of the 41st Meeting of the Association for Computational Linguistics, 2003.

[10] Lami, G., Gnesi, S., Fabbrini, F., Fusani, M., and Trentanni, G., “An Automatic Tool for the Analysis of Natural Language Requirements,” published as Technical Report 2004-TR-40, Consiglio Nazionale delle Ricerche, Instituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2004.

[11] Landis, J.R. and Koch, G.G., “The measurement of observer agreement for categorical data,” Biometrics, 33, 1977, pp. 159-174.

[12] Letier, E., Kramer, J., Magee, J. and Uchitel, S., "Monitoring and Control in Scenario-Based Requirements Analysis," Proceedings ICSE 2005 - 27th International Conference on Software Engineering, ACM Press, St. Louis, Missouri, USA, May 2005.

[13] Lin, D., “Dependency-based Evaluation of MINIPAR,” In Workshop on the Evaluation of Parsing Systems, Granada, Spain, May, 1998.

[14] Lu, R., Jin, Z., and Wan, R., “Requirement Specification in Pseudo-Natural Language in PROMIS,” In proceedings of 19th International Computer Software and Applications Conference (COMPSAC'95), pp. 96-101, 1995.

[15] Meyer, B., “On Formalism in Specifications,” IEEE Software, 2(1): pp. 6-26, January 1985.

[16] Osborne, M. and MacNish, C.K., “Processing natural language software requirement specifications,” In Proceedings of ICRE'96: 2nd IEEE International Conference on Requirements Engineering, pp. 229-236. IEEE Press, 1996.

[17] Rolland, C. and Proix, C., “A Natural Language Approach For Requirements Engineering,” Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering, vol. 593 of Lecture Notes in Computer Science, pp. 257-277, Manchester, United Kingdom, 1992.

[18] Wilson, W., “Writing Effective Requirements Specifications," USAF Software Technology Conference, Utah, 1997.

[19] Wilson, W., Rosenberg, L. and Hyatt, L., “Automated Quality Analysis of Natural Language Requirement Specifications,” 14th Annual Pacific Northwest Software Quality Conference, Portland, 1996.Bowman, B., Debray, S. K., and Peterson, L. L. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst., 15,* 5 (Nov. 1993), 795-825.