
Publications | Leila
Machine Learning
Collective Agreem
Full Professor Doss
Automatic Quality
ieeexplore.ieee.org/document/4385497/
67%
ents in SRS documents
Most Visited
Google
Calendar
MyConcordia Home p...
My Drive - Google Dri...
Class Schedule & Regi...
École secondaire Mon...
access to IEEE Xplore for your organization?
Browse Conferences > Quality Software, 2007. QSIC ...
Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier
Sign In or Purchase to View Full Text
6 Paper Citations
247 Full Text Views
Related Articles
FTS: a high-performance CORBA fault-tolerance service
Schedulability in model-based software development for distributed real-time sys...
View All
3 Author(s)
Ishrar Hussain ; Olga Ormandjieva ; Leila Kosseim
View All Authors
Abstract
Authors
Figures
References
Citations
Keywords
Metrics
Media
Abstract:
The success of a software project is largely dependent upon the quality of the Software Requirements Specification (SRS) document, which serves as a medium to communicate user requirements to the technical personnel responsible for developing the software. This paper addresses the problem of providing automated assistance for assessing the quality of textual requirements from an innovative point of view, namely through the use of a decision- tree-based text classifier, equipped with Natural Language Processing (NLP) tools. The objective is to apply the text classification technique to build a system for the automatic detection of ambiguity in SRS text based on the quality indicators defined in the quality model proposed in this paper. We believe that, with proper training, such a text classification system will prove to be of immense benefit in assessing SRS quality. To the authors' best knowledge, ours is the first documented attempt to apply the text classification technique for assessing the quality of software documents.
Published in: Quality Software, 2007. QSIC '07. Seventh International Conference on

Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier

Ishrar Hussain, Olga Ormandjieva and Leila Kosseim
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
{h_hussa,kosseim, ormandj}@cse.concordia.ca

Abstract

The success of a software project is largely dependent upon the quality of the Software Requirements Specification (SRS) document, which serves as a medium to communicate user requirements to the technical personnel responsible for developing the software. This paper addresses the problem of providing automated assistance for assessing the quality of textual requirements from an innovative point of view, namely through the use of a decision-tree-based text classifier, equipped with Natural Language Processing (NLP) tools. The objective is to apply the text classification technique to build a system for the automatic detection of ambiguity in SRS text based on the quality indicators defined in the quality model proposed in this paper. We believe that, with proper training, such a text classification system will prove to be of immense benefit in assessing SRS quality. To the authors' best knowledge, ours is the first documented attempt to apply the text classification technique for assessing the quality of software documents.

1. Introduction

Requirements Engineering (RE) is concerned with the gathering, analyzing, specifying and validating of user requirements that are documented in natural language for the most part. Software Requirements Specification (SRS) is one of the most important artifacts produced during the software development lifecycle. The success of a software project is largely dependent upon the quality of SRS documentation, which serves as an input to the design, coding and testing phases. The quality assessment of SRS documents often takes 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, each with the potential to be misinterpreted. Consequently, even though manual inspection of document quality is the most common method used, it is also one of the costliest phases of RE.

Research Objectives: The aim of our research is to provide automated assistance for assessing the quality of SRS text. The work presented in this paper is the first step toward a larger project aimed at applying NLP techniques to the RE process (see Figure 1).

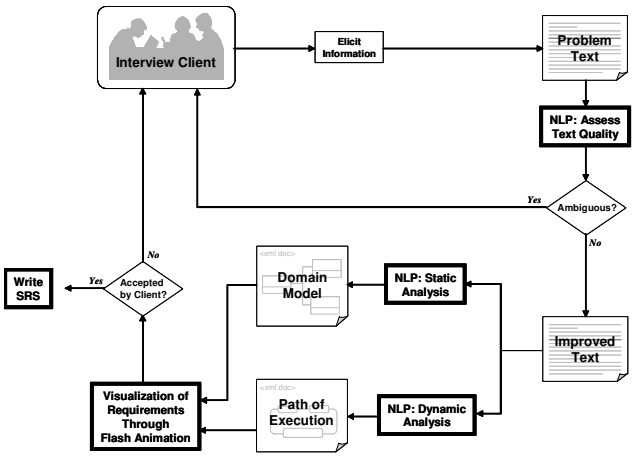

Figure 1. NLP-Based Quality Assessment in Requirements Engineering

The objective of NLP assessment in the context of that project can be expressed in terms of three main goals:

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

G2. This involves 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. This involves graphical visualization and animation of the conceptual models extracted from the requirements text for the user's validation and feedback.

The objective of this paper is to report on the first step toward the automation of quality evaluation in SRS documents (see G1 above) by means of a text classification system. The primary goals are to identify the textual ambiguities in the requirements elicitation phase before the conceptual modeling of the requirements begins, and to automate the detection of ambiguity in SRS documents in terms of reading comprehension. Our research aim, as a "proof of concept", was to build a text classification system and test its applicability for detecting ambiguity in SRS documents.

Quality of SRS text: The notion of SRS quality is derived from the existing guidelines in the literature for writing SRS documentation (such as [7, 9, 13, 19, 26, and 27]) and from the authors' experience. Some SRS quality characteristics are outlined in the IEEE Standard 830-1998 [6], which describes the best practices recommended by the IEEE for writing an SRS document, and defines the quality characteristics of a "good" SRS document. They are: (1) Correct, (2) Unambiguous, (3) Complete, (4) Consistent, (5) Ranked for importance, (6) Verifiable, (7) Modifiable and (8) Traceable. The definition of "unambiguous" set by the standard corresponds to an SRS document in which each of its statements has only one interpretation. The IEEE standard further mentions that the inherently ambiguous nature of NL can make the text of an SRS document fail to comply with the above rule, making it ambiguous, and thereby degrading the overall quality of the document.

Research hypothesis (1): 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.

We have applied a goal-oriented approach to define the hierarchical decomposition of the ambiguity concept into quantifiable characteristics (indicators) affecting the quality of the text from different viewpoints. The NLP text classification technique was employed to build a system for the automatic detection of ambiguity in requirements documents in terms of the quality indicators defined in the quality model.

Research hypothesis (2): We understand that none of the previous work has tested the applicability or performance of using a text classification system to automate the detection of textual ambiguities in SRS documents. Thus, our second research hypothesis was that a decision-tree-based text classifier, trained with and tested against human annotated samples, can best emulate human decisions taken while detecting ambiguity in SRS documents.

To the authors' knowledge, ours is the first documented attempt to apply the NLP text classification technique to software requirements quality assessment. We believe that the research results reported in this paper not only prove our concept for practical use, but also introduce some profound analyses of this topic that can fuel further investigations among the research community.

The paper is organized as follows: Section 2 discusses various viewpoints regarding the quality of SRS textual documentation. The underlying quality model is introduced in section 3. Our text classification system and its performance analysis are presented in section 4. A critique of our research results in comparison to related work is given in section 5. Finally, our conclusions and guidelines for future work are outlined in section 6.

2. Views on Quality of SRS Text

The research described in this paper is concerned with the challenges inherent in the reading comprehension of the initial textual requirements (see G1, section 1). Reading comprehension can typically be analyzed from two broad viewpoints: the literal meaning of a text (or surface understanding) and its interpretation (or conceptual understanding). In the context of our work, we use the term "surface understanding" to denote 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. By contrast, we use the term "conceptual understanding" to denote how much a developer or a requirements engineer would gain in the analysis and specification of a system by carefully reading/examining its problem texts only. The increasing difficulty at this level of understanding represents possible failure to identify the precise meaning of the text (interpretation), which is also referred to in Meyer's work [19] and in the IEEE Standard [6] and is defined as *ambiguity*.

In our work, we use the term "ambiguity" in its more general sense—the characteristics of the language used in an SRS document that make its content difficult to interpret or to realize in the

software development process. Ambiguity can exist in different forms within the NL texts of an SRS document. Consider the following two examples:

(1) *Most commonly, the orders are matched in price/time priority: The order with the better price has a higher priority than an order with a worse price* [15].

(2) *Design a program that allows a network operator to plan changes in every parameter of every cell in the network. These planned changes should be verified for consistency and correctness and then applied to the network in the least disturbing way* [12].

In example (1), it is impossible to evaluate what is meant by a "better" price or a "worse" price, since such interpretations are highly subjective. Moreover, the text does not explain by how much the "priority" will change with an increase or decrease in price. Thus, example (1) is ambiguous, since we cannot deduce its true meaning based on this text. Example (1) demonstrates ambiguity at the level of surface understanding. Several surface factors can be involved at this level, e.g. the length of sentences, ambiguous adjectives and adverbs, passive verbs, etc.

In example (2), the text does not specify how to verify the changes to the cells for consistency or correctness. So, we find example (2) to be ambiguous as well, since the feature it describes cannot be realistically implemented in an analysis model, let alone design or executable code, due to lack of information. In [19], Meyer defined such a phenomenon as "silence". In his paper [19], he details the areas of natural language (NL) where a human specifier is most prone to making mistakes. His comprehensive study presents a thorough description of such mistakes by classifying them into seven distinct categories: *silence*, *over-specification*, *contradiction*, *ambiguity*, *forward reference* and *wishful thinking*. Note that, like the "silence" shown in example (2), each of these seven sins can make the text so ambiguous that it cannot be realized in any analysis model.

Another point of view regarding reading comprehension deals with the impact the *scope* (sentence or passage) of ambiguity has on the overall understanding of the text. For instance, our example (2) demonstrates ambiguity at the conceptual understanding level, because it suffers from a lack of information (silence). Such ambiguity usually exists within a passage comprising more than one sentence or paragraph. Thus, a single sentence may not be enough to identify the text as ambiguous. By contrast, the kind of ambiguity introduced in example (1) is limited to the scope of a single sentence.

Stakeholders involved in the use and development of a system should be able to understand an SRS at both the surface and conceptual levels. Thus, writing SRS documents which are unambiguous, both at the surface level and at the conceptual level, is critical in the software lifecycle. If not detected early, such ambiguities can lead to misinterpretations in the course of requirements engineering or at a subsequent phase of the software development lifecycle, causing an escalation in the cost of software development. The research reported in this paper targets an automatic assessment of SRS text by means of a classifier to automatically flag ambiguous and unambiguous texts at both the surface and conceptual levels within their corresponding scopes at an early stage of the requirements elicitation process, which can save a great deal of aggravation, not to mention cost. For this purpose, we had to identify the discriminating features of the text that characterize the quality of an SRS text from different viewpoints. The next section introduces the underlying hierarchical quality model which depicts the decomposition of ambiguity into measurable features compiled directly from the text.

3. Underlying Quality Model

The challenge in defining a quality model for SRS text originates in the subjective nature of NL ambiguity. To surmount this, we have applied a goal-oriented approach to address objective and discriminating linguistic features upon which both surface and conceptual levels of ambiguity depend. For this purpose, we have developed separate specific guidelines and examples of what is to be considered ambiguous in terms of surface and conceptual understanding respectively. The guidelines indicate what to look for in a text, but do not give any strict instructions. They were developed by taking into account previous work in the field, such as [7, 9, 13, 19, 26, and 27]. For example, in terms of surface understanding, the following phenomena could render a passage ambiguous:

- *Long sentences. A long sentence can be identified by many commas or semi-colons, several main verbs or clauses in a sentence, the use of parentheses, etc.*
- *Introduction of many concepts in a single sentence.*
- *Grammatical errors or heavy syntactical constructions.*
- *Numerous adjectives and adverbs.*
- *Numerous verbs in the passive voice.*

In terms of conceptual understanding, Meyer's seven sins [13] were considered as indicators of

ambiguity. By contrast, if a directive phrase was found in a passage, e.g. “for example”, “note:”, “e.g.”, “such as”, “(IMAGE)”, “(TABLE)”, it would render the passage less ambiguous.

Our initial quality model reported in [20] considered both surface and conceptual understanding indicators. Conversely, the features influencing ambiguity in conceptual understanding of the SRS documents required domain knowledge and deeper semantic analysis, for which available NLP tools are inefficient. We, therefore, focused our objective on developing the system for detecting ambiguity solely at the level of surface understanding; the corresponding quality model is given in Figure 2.

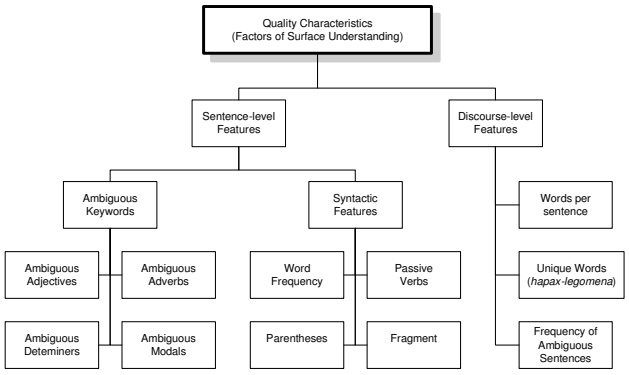

Figure 2. Underlying Quality Model

Ideally, assessing the linguistic quality of an SRS document should be performed automatically, or at least assisted in that task by automated means. In [20], we have presented a study investigating the feasibility of an annotated tool for automatic assessment of quality in SRS documents. The results were 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.

4. Decision-Tree-Based Text Classifier

Our work targets the automatic assessment of textual requirements in terms of their ambiguity by means of a text classification system to actually flag ambiguous and unambiguous texts.

Text classification is a technique to classify text contents or documents into two or many groups based on different characteristics. It is being used for e-mail classification and spam detection [11], clustering and organizing documents [14], information extraction [24] and retrieval [10, 29] etc. A text classifier generally uses an implementation of a machine learning algorithm. In a supervised training method, values of discriminating features of the classes, which the classifier uses to classify documents, are collected from a large number of documents. Then, with the aid of the machine learning algorithm, the classifier is ‘trained’ based on the feature values of the documents, which belong to all the different classes. Thus, when a new unclassified document is supplied to the classifier, it is able to classify the document into a class based on its feature values.

To build our classifier we needed an annotated corpus for the purpose of training and testing.

4.1. Annotated Corpus

To our knowledge, no standard corpus of SRS documents has been collected for this task. We therefore built our own for the annotation process extracted from 25 different problem descriptions (each from a different domain) related to SRS. The problem descriptions were collected from the ACM’s OOPSLA Designfest, available online at http://designfest.acm.org/. Annotating an entire problem description of several pages, would be difficult and not particularly informative if we later wish to identify specific features for an automatic classification task. We therefore chose to split up our collection of problem descriptions into smaller segments that derived two different sets of corpora: the *discourse-level corpus* and the *sentence-level corpus*. For our discourse-level corpus, we extracted 165 passages from our collection of 25 problem descriptions, where each passage contained 140.22 words on average. Our sentence-level corpus consisted of 1211 sentences, all extracted from our collection of 25 problem descriptions. Each of the extracted sentences contained 19.1 words on average.

To perform the manual classification, we asked four annotators (two professors and two graduate students) to annotate the samples as “Ambiguous” or “Unambiguous” in terms of surface understanding and conceptual understanding. All the annotators had software engineering backgrounds, but from different fields of computer science with significant exposure to dealing with SRS documents.

All the passages and sentences of the corpora were annotated into two distinct categories — “ambiguous” and “unambiguous”, using a standard scoring procedure and employing a median voter model [3]. The corpora can be used for the purpose of training and/or testing future text classifiers in this field. Figure 3 illustrates the steps of our corpus annotation process. The details on the annotation methodology and the analysis of the reliability of the annotations are discussed in [20]. The results of the inter-annotator

agreement and the reliability of the annotations were sufficient to lead us to believe that an automatic system can be built to emulate the decision-making process of human annotators, and to automatically classify SRS documents.

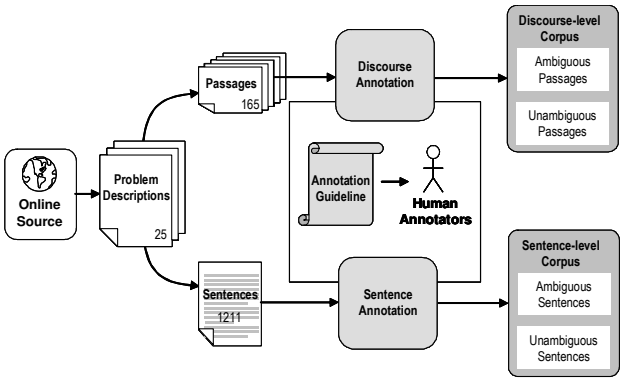

Figure 3. Corpus Annotation

In addition, analysis of the above-mentioned results indicated 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 confirmed our premise 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.

4.2. Features Extraction

We have successfully devised NLP-driven feature extractor tools that extract discriminating features of the SRS text influencing ambiguity in surface understanding. The tools can output data files (in standard ARFF format), each representing an annotated corpus in terms of feature vectors. The file can be used to train any machine learning algorithm to develop a classifier.

4.3. Algorithm

We chose the C4.5 decision-tree learning algorithm [22] 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. For our next experiment, we used the machine learning workbench, called Weka [28] that provided a Java-based implementation of the C4.5 (revision 8) algorithm, along with the necessary framework for training and evaluating our classifier.

We developed a decision-tree-based text classifier that works at both sentence and discourse levels of an SRS text and detects ambiguity in surface understanding (see sections 4.4 and 4.5). The classifier can also be trained with new training data, which makes it robust in dealing with unseen samples of SRS in future. The classifier attained an impressive accuracy, high enough to be applicable in a practical semi-automated environment. This eventually proves our research hypothesis (1) of the usability of a text classifier in detecting ambiguity in SRS documents. The details are given below.

4.4. The Sentence Classifier

We have developed the sentence classifier using the C4.5 decision-tree learning algorithm (see section 4.3). The classifier has a training module, which can be trained using the sentence level training data and can dynamically generate a decision-tree based on the data. This decision-tree-based classifier automatically classifies a sentence into ambiguous or unambiguous in terms of surface understanding.

Our sentence-level *Feature Extractor* tool extracts features required for detecting ambiguity in a sentence. It counts the frequency of occurrences of valid features and ambiguous keywords only in each of the sentences of our corpus and stores them in the training data file. The training data file also holds the corresponding annotation of each sentence. Figure 4 illustrates the steps of the classification:

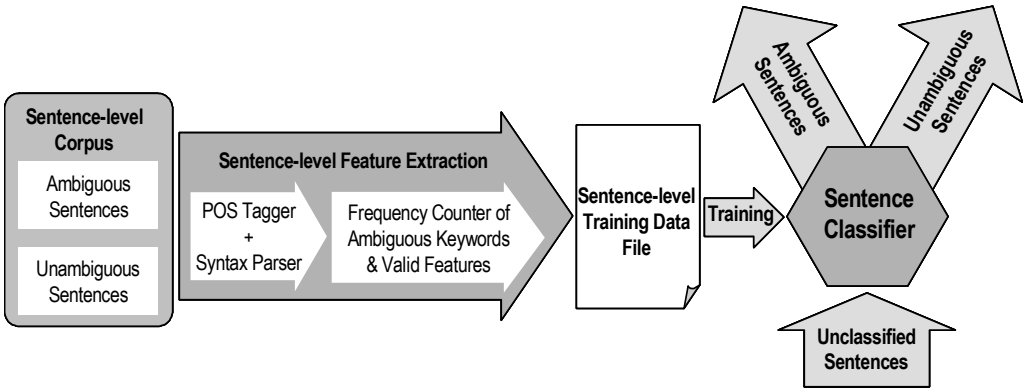

Figure 4. Steps of Sentence-level Classification

We experimented with the sentence classifier by training the classifier with different combinations of sentence level features and testing its performance, as compared to that of human annotation. The results revealed high performance accuracy, in sufficient degree for the classifier to be applicable in practical fields (see section 4.6).

Our results with this initial experiment affirm that the task of detecting ambiguity in terms of surface understanding is indeed achievable by means of currently available NLP tools and text classification techniques. The accuracy of our sentence classifier establishes its applicability in practical fields, where ambiguity is detected at the sentence level. But, since our primary concern was to detect ambiguity at the discourse level, we continued with our work of building the ultimate classifier that could best emulate the decisions of our human annotators by classifying a discourse based on its ambiguity at the level of surface understanding.

4.5. The Discourse Classifier

We built the discourse-level classifier using our sentence classifier. Figure 5 illustrates the process. Here, the sentence classifier was used to count the number of ambiguous sentences in each of the instances (passages) in our corpora, and classify an instance based on the density of ambiguous sentences (along with two other discourse features, which were ignored by our classification algorithm on the basis of our training data file) and the corresponding annotation of the instance.

We then experimented with the discourse classifier by training it with different discourse level features, and testing its performance against that of human annotation. The results show high performance accuracy, affirming that the approach of using a text classifier is applicable in practical fields for detecting ambiguous passages in SRS documents (see section 4.6).

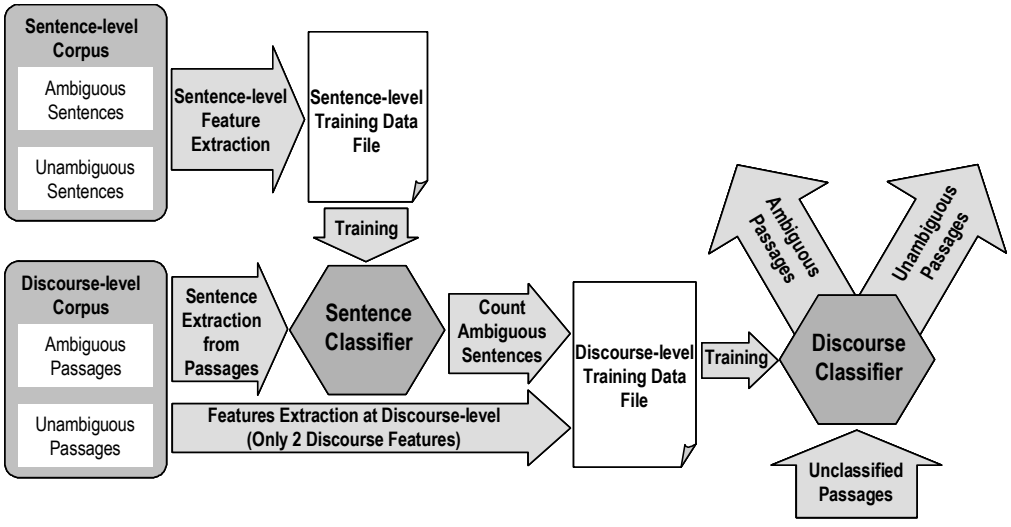

Figure 5. Discourse-level Classification using the Sentence Classifier

4.6 Performance Evaluation

The performance of a system can be evaluated from different perspectives. For evaluating the performance of our system we chose to compare the performance of our discourse-level classifier to the human performance.

For this purpose, we compared the level of agreement of the classifier and that of each of the annotators with the gold standard (decided by the annotations of *Annotator3*, see [20] for more details). Figure 6 shows that the classifier emulates the annotations of *Annotator3* successfully by fitting its decision-tree properly with the annotator's annotation and attaining a high "substantial" level of agreement when training and testing were done on the same corpus, and also when 10-fold-cross-validation was performed (Cohen's [2] kappa index 0.7572 and 0.7203, respectively).

This result is also considerably higher than the average level of agreement human annotators had with *Annotator3* (Kappa index 0.5426 on average, denoting a "moderate" level of agreement). As the discourse-level classifier system aims to be a part of the requirements elicitation and analysis process, where its outputs are expected to be verified by an experienced requirements analyst, we can endorse its performance to be fully adequate for the purpose.

The above results of the performance analysis thoroughly prove our research hypothesis (2) affirming that the approach of using a text classifier is applicable in practical fields for detecting ambiguous passages in SRS documents. Although our work is fully concentrated in detecting ambiguities at the level of surface understanding, it will facilitate our future work of detecting conceptual ambiguities by improving the quality of the SRS text.

The classifier can also be trained with new training data, which makes it robust to deal with unseen samples of SRS in future. The classifier attained an impressive accuracy, high enough to be applicable in a practical semi-automated environment. This ultimately proves our hypothesis of the usability of a text classifier in detecting ambiguity in SRS documents.

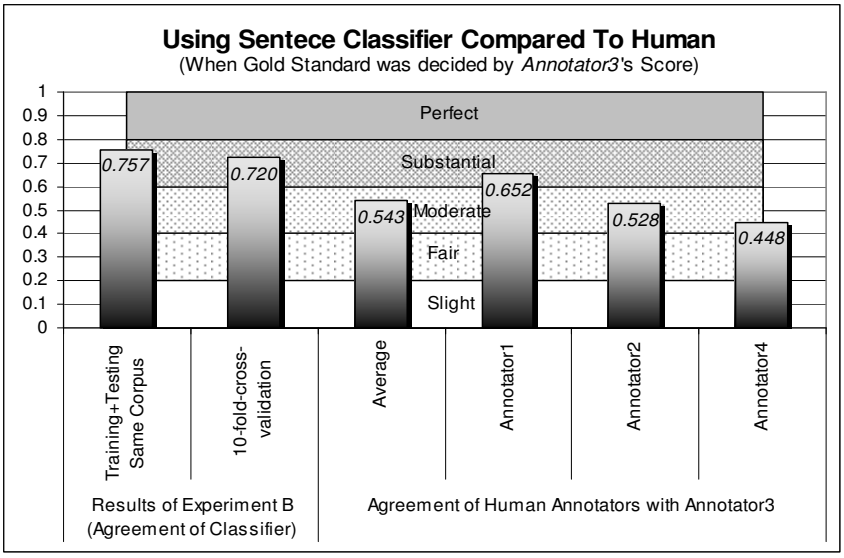

Figure 6. Performance Evaluation of Discourse-level Classifier that uses the Sentence Classifier Compared to Human Annotators in Terms of Their Level of Agreement with Annotator3

4.7 Prototype

A small prototype (written in Java) has been developed to demonstrate the use of our classifier. We used the annotated corpus of this study to train the text classification prototype that can automatically classify SRS documents based on their probable defects at the level of surface understanding only. The system can label a discourse as ambiguous, and, at the same time, identify the sentences that are responsible for making it so.

The prototype named Requirements Specification Ambiguity Checker (ReqSAC) is platform independent, and can be executed online, or from within the Rational XDE environment. This ensures future usability of the system. The UI also features on-the-fly text editing to correct errors and enhance the quality of the SRS, using the same linguistic features that our annotators used to detect errors at the level of

surface understanding. Figure 7 shows a snapshot of the main window of the prototype.

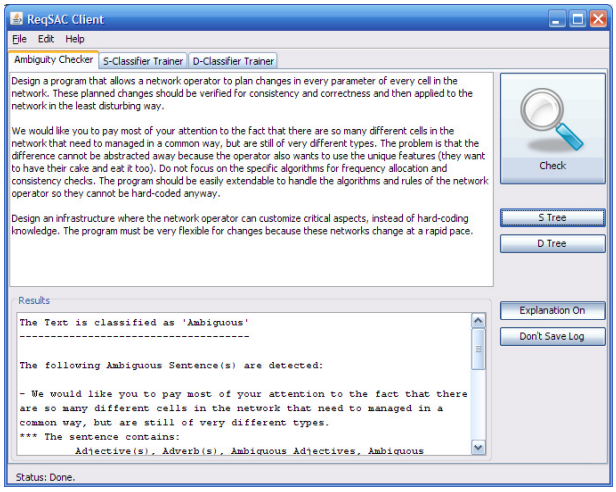

Figure 7. A Snapshot of the ReqSAC Client

5. 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 [19], 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*. [9] 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*. [16] 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 errors extremely difficult for a user to detect from the model at a later point.

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, 18], but this is not truly NL. Others propose restricting the grammar to consider only a subset of NL when writing requirements specification [5, 8, 23, and 25]. 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 accuracy, which means that they can be used for real-world texts. Osborne *et al*. [21], 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, that of Wilson *et al*. [26, 27], 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 reveal 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*. [7] and Gnesi *et al*. [13] address the issue by proposing a tool called "QuARS: Quality Analyzer for Requirements Specification". QuARS syntactically parses the sentences using the MINIPAR parser [17], 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 a list of handpicked specific keywords, rather than on more general classes of words. At every stage of processing, QuARS requires the use of a different "modifiable" keyword 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 work provides a benchmark for evaluation of 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, and an upper bound on what we can expect automatic tools to achieve.

6. Conclusions and Future Work

This paper addresses the problem of detecting ambiguities in SRS documents. Acknowledging the fact that none of the previous work has tested the applicability or performance of using a text classification system to automate such detection process, the research results proved our hypothesis by affirming that the approach of using a text classifier is applicable in practical fields for detecting ambiguous passages in SRS documents. The work also encompassed some important related topics, e.g. how difficult it is to manually detect ambiguity in SRS documents and how the automatic tools developed can compare to human performance.

To prove our concept, we developed a text classification system that can detect ambiguity in an SRS document by classifying its passages as ambiguous or unambiguous. The system yields high performance accuracy demonstrating 86.67% accuracy under the critical conditions of using 10-fold-cross-validation technique [20]. When comparing whether its results agreed with the decisions of an expert, it was found to outperform human annotators with *average* expertise in detecting ambiguities. It can also be affirmed that the system will perform better in practical fields with the inclusion of new training data. We also built a prototype of this system to demonstrate its practical usage.

Our work strictly concentrated on detecting surface understanding ambiguities, while avoiding the detection of ambiguities which are at the level of conceptual understanding. The reason behind this was that to date, efficient resources capable of performing semantic analysis or holding domain knowledge, required for detecting the presence of the features in text that degrades conceptual understanding of an SRS document, have been unavailable. This also re-affirmed the claims of Meyer [19] and Kamsties *et al* [9] that ambiguities of SRS text may not be detected without the aid of formalism. Thus, we proposed that our NLP-driven quality assessment project would deal with detecting ambiguity at the level of conceptual understanding by introducing a level of formalism (with domain model generation and simulating the path of execution) and following a semi-automated approach with continual feedback of the client (see Figure 1). The reported research results, which are part of a larger project (see section 1), successfully establish its approach to detecting ambiguities that are at the level of surface understanding, and, thus, facilitating the processes of semi-automated analysis, to be performed by upcoming modules. We strongly believe our system, with the potential to clean up ambiguities at the level of surface understanding, will not only serve the aforesaid project, but also be useful as a standalone application working in conjunction with the requirements specification writing tools. The prototype of this system is, therefore, implemented to run both as a standalone application and from within the Rational XDE environment.

Our future work will focus on introducing more training data to improve its efficiency in dealing with unseen SRS texts, and on including a collocation extractor with our keyword generation tool. Finally, we look forward to its successful integration with the module that detects ambiguity in conceptual understanding.

8. References

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

[2] Cohen, J. (1960). "A coefficient of agreement for nominal scales". *Educational and Psychological Measurement, 20*, pp. 37–46.

[3] Congleton, R.D. (2004). "The Median Voter Model". *Encylopedia of Public Choice: Volume II (eds. Charles. K. Rowley and Friedrich Schneider)*, pp. 382–387.

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

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

[6] The Institute of Electrical and Electronics Engineers, Inc. (1998). *IEEE recommended practice for software requirements specifications (IEEE Std 830-1998)*, October, New York.

[7] Fabbrini, F., Fusani, M., Gnesi, S. & Lami, G. (2001). “An Automatic Quality Evaluation for Natural Language Requirements”. In *Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'01*, Interlaken, June pp. 4–5.

[8] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M. & Moreschini, P. (1994) “Assisting requirement formalization by means of natural language translation”. *Formal Methods in System Design, vol. 4*, 243–263.

[9] Kamsties, E., Berry, D.M. & Paech, B. (2001). “Detecting Ambiguities in Requirements Documents Using Inspections”. In *Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)*, July, Paris, 68–80.

[10] Kleinberg, J. (1999). “Authoritative sources in hyperlinked environment”. *Journal of the ACM, 46(5)*, 604–632.

[11] Kolcz, A. & Alspector, J. (2001) “SVM-based filtering of e-mail spam with content-specific misclassification costs”. In *Workshop on Text Mining, IEEE ICDM-2001*, IEEE Press, Piscataway, NJ.

[12] Kriens, P. (1996). CellKeeper, a Cellular Network Manager. OOPSLA DesignFest®, 1996. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/ CellKeeper/CellKeeper_96.htm

[13] Gnesi, S., Lami, G., Trentanni, G., Fabbrini, F., & Fusani, M. (2005). “An Automatic Tool for the Analysis of Natural Language Requirements”. *International Journal of Computer Systems Science and Engineering, Special issue on Automated Tools for Requirements Engineering, 20(1)*, Leicester, UK: CRL Publishing.

[14] Larsen, B. & Aone, C. (1999). “Fast and effective text mining using linear-time document clustering”. Proceedings of ACM SIGKDD-1999, pp. 16–22, August, San Diego.

[15] Layda, T. (2000). Order Matcher for an Electronic Stock Market. OOPSLA DesignFest®. Last retrieved on July 18, 2007 from designfest.acm.org/Problems/ OrderMatcher/ OrderMatcher_00.htm

[16] Letier, E., Kramer, J., Magee, J. & Uchitel, S. (2005). “Monitoring and Control in Scenario-Based Requirements Analysis”. Proceedings the 27th International Conference on Software Engineering, St. Louis.

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

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

[19] Meyer, B. (1985) “On Formalism in Specifications”. *IEEE Software, 2(1)*, January 1985, 6–26.

[20]] Hussain, I. (2007). “Using Text Classification System to Automate Ambiguity Detection in SRS Documents”. *Master’s Thesis*, Computer Science and Software Engineering Department, Concordia University, Montreal, Canada, August.

[21] Osborne, M. & MacNish, C.K. (1996) “Processing natural language software requirement specifications”. Proceedings of ICRE'96: 2nd IEEE International Conference on Requirements Engineering, IEEE Press, 229–236.

[22] Quinlan, J.R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

[23] Rolland, C. & Proix, C. (1992). “A Natural Language Approach For Requirements Engineering”. Proceedings of the 4th International Conference CAiSE'92 on Advanced Information Systems Engineering, LNCS 593, Manchester, UK, pp. 257–277.

[24] Spertus, E. (1997). “Smokey: Automatic recognition of hostile messages”. Proceedings of IAAI-97, Providence, July, pp. 1058–1065.

[25] Tjong, S.F., Hallam, N. & Hartley, M. (2006). “Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns”. Proceedings of the 6th IEEE International Conference on Computer and Information Technology, (CIT '06), September,.

[26] Wilson, W. (1997) “Writing Effective Requirements Specifications”. In USAF Software Technology Conference, Utah, April.

[27] Wilson, W., Rosenberg, L. & Hyatt, L. (1996) “Automated Quality Analysis of Natural Language Requirement Specifications”. Proceedings of the 14th Annual Pacific Northwest Software Quality Conference, Portland.

[28] Witten, I. H. & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques (2nd ed.)*. San Francisco, CA: Morgan Kaufman.

[29] Zhang, Z. & Varadarajan, B. (2006) “Utility scoring of product reviews”. Proceedings of ACM CIKM-2006, Arlington, pp. 51–57.