An Automatic Tool for the Analysis of Natural Language Requirements

Stefania Gnesi, Giuseppe Lami, Gianluca Trentanni
Fabrizio Fabbrini, Mario Fusani,
Istituto di Scienza e Tecnologia dell’Informazione “A. Faedo”
Via Moruzzi, 1 – I56124 Pisa, Italy
{gnesi, glami, trentanni, fabbrini, fusani}@isti.cnr.it

Abstract

Using automatic tools for the quality analysis of Natural Language (NL) requirements is recognized as a key factor for achieving software quality. Unfortunately few tools and techniques for the NL requirements analysis are currently available. This paper presents a methodology and a tool (called QuARS - Quality Analyzer for Requirement Specifications) for analyzing NL requirements in a systematic and automatic way. QuARS allows requirements engineers to perform an initial parsing of the requirements in order to automatically detect potential linguistic defects that could cause interpretation problems at subsequent stages in developing the software. This tool is also able to partially support the consistency and completeness analysis by clustering the requirements according to specific topics.

1. Introduction

Producing high quality software requirements is the first step towards software quality. It is well known that inaccuracies in requirements documents can lead to serious problems in all the subsequent software development phases. Using methods and tools to analyse software requirements may improve the effectiveness of the requirement process and the quality of the final product.

The analysis of software requirements in terms of ambiguity, inconsistency and lack of correctness has been extensively exploited in recent years. A way to deal with such problems is through their description in some semiformal/formal languages. Graphical object modelling languages have become very popular including, for example, the semiformal language UML [22]. On the other hand, formal specification languages (such as Z [24], B [1], LOTOS [4], etc.) have been defined specifically to add formality and remove ambiguities. However, they are difficult to understand by non-experts and this limits their practical application to some restricted domains. In practice, NL is still the most common way to express software requirements.

The use of NL for specifying requirements, despite its inherent ambiguity, does have some advantages such as the ease with which they can be shared among the various people involved in the software development process. In fact, an NL requirements document can be used in several development phases of the final product. For example, it can be used as a working document as an input for system architecture designers, testers, and user manual editors. Moreover, it can also be used to establish an agreement between customers and suppliers, and as a data source for project management.

Not many techniques for NL requirements analysis exist, and only few of them are supported by automatic tools for their application. The following list, while not exhaustive, includes the most popular means and practices used for mitigating the problems due to the use of NL in requirements specifications:

- Means for expressing NL requirements: in practice several means and techniques have been defined in order to mitigate the inherent ambiguity of NL. The most common ones are based on the use of templates for structuring requirements documents or the adoption of a restricted English (avoiding ambiguous terms and styles) for expressing the requirements. Following these principles, the adoption of Use Cases [5] as a notation to express requirements in industrial projects is an interesting approach. Use Cases allow a functional description of the requirements and impose a (light) formalism based on the NL to the requirements.

- Practices for mitigating the effects of the inherent ambiguity in NL: because no technique or tool can guarantee the absence of ambiguities in NL requirements, some countermeasures are frequently adopted. For example, joint reviews of requirements documents are carried out together by customers and suppliers to check that the various developers and customers have the same understanding of each requirement. Glossaries are often of great help in doing such reviews.

In the software industry, ambiguity analysis, consistency and completeness verification of software requirements is usually performed by humans, generally through a tedious procedure of reading requirements documents and looking for linguistic errors.
Some recent market research [19] on the potential demand for linguistic automatic tools for requirements analysis concluded: “Because an engineering approach suggests the use of linguistic tools suited to the language employed in the narrative description of user requirements, we find that in a majority of cases it is necessary to use NL Processing systems capable of analysing documents in full natural language”.

This paper provides a contribution to meet the demand for quality analysis tools for NL requirements, i.e. the automatic detection of ambiguous, incomplete, and contradictory NL requirements. In previous works [9, 10] we defined a method for supporting the analysis of NL requirements in order to extract information from and identify defects in their representation. On the basis of this method, we have developed QuARS (Quality Analyzer for Requirements Specifications), an innovative tool for automatically analysing NL requirements. QuARS performs a lexical and syntactical parsing of software requirements expressed in NL, and identifies defective sentences in requirements documents so as to reveal ambiguities and understand problems in them.

The paper is structured as follows: Section 2 presents a survey of the works related to the NL requirements analysis. Section 3 outlines our method for the analysis of NL requirements. In Section 4 the tool that automatizes this methodology is presented and in Section 5 its components are described. In Section 6 some industrial case studies using the tool are reported and discussed. Section 7 shows how the tool can handle non-functional requirements. Finally, in Section 8 conclusions are provided and future works described.

2. Related Works

Several studies on quality in NL requirement documents can be found in the literature, below we overview those that are particularly relevant to our research.

Macias and Pulman [18] apply domain-independent Natural Language Processing (NLP) techniques to control the production of NL requirements. They apply NLP techniques to requirements documents in order to check the vocabulary used, which must be fixed and agreed upon, and the style of writing, i.e. a set of pre-determined rules that should be satisfied in order to make documents clear and simple to understand. They associate an ambiguity rate with sentences, depending on the degree of syntactic and semantic uncertainty in the sentence. The information is conveyed by discovering under-specifications, missing information and unconnected statements. Finally, they discuss how NLP techniques can help in the design of subsets of English grammar to limit the generation of ambiguous statements.

Goldin and Berry [13] implemented a tool for the extraction of abstractions from NL texts, i.e. of repeated segments identifying significant concepts on the application field of the problem at hand. The technique proposed is limited to a strict lexical analysis of the text.

Fuchs [12] suggests solving the problems related to using NL in requirements documents by defining a limited natural language, called Attempt Controlled English (ACE), which can be easily understood by stakeholders and by anyone involved in the software development process. This subset of English is simple enough to avoid ambiguities, so that domain specialists are allowed to express requirements using NL expressions and to combine these with the rigor of formal specification languages.

Kamsties and Paech [17] focus on the ambiguity evaluation of NL requirements. Their premise is that ambiguity in requirements is not just a linguistic-specific problem and suggest using checklists that cover not only linguistic ambiguity but also domain-specific ambiguity.

Mich and Garigliano [20] propose a set of metrics for syntactic and semantic ambiguity in requirements. Their approach is based on the use of information on the possible meanings and roles of the words within a sentence and on the possible interpretation of a sentence. This is done using the functionalities of a tool called LOLITA.

Fioresmith [11] provides an as yet not exhaustive list of the characteristics that good-quality requirements should have, along with a discussion of the typical defects in requirements.

Natt och Dag et al. [21] recently presented an approach based on statistical techniques for the similarity analysis of NL requirements aimed at identifying duplicate requirement pairs. This technique may be successfully used for revealing interdependencies and may then be used as a support for the consistency analysis of NL requirements. In fact, the automatic determination of clusters of requirements dealing with the same arguments may support human analysis in detecting inconsistencies and discrepancies by focusing on smaller sets of requirements.

The CREWS (Cooperative Requirements Engineering With Scenarios) [3] project aims to define a scenario-based method to elicit and validate requirements. This method is based on the cooperation between users and requirements engineers in defining scenarios. Scenarios are considered as a way to express the goal in a format acceptable by both parties. In particular, the CREWS-L’Ecrivain defines an approach where the scenarios are expressed in a middle
ground between completely free mode of use of NL and predefined templates, and it combines the use of informal narrative prose to express scenarios with structured NL to analyze them.

Ambriola and Gervasi [2] aim to achieve high quality NL requirements through CIRCE, a system that can build (semi-) formal models almost automatically. Their system does this by extracting information from the NL text of the requirements, and then measures and checks the consistency of these models. CIRCE promotes the use of a suitable style in the requirements.

Below the above studies are classified into three categories with their main advantages and drawbacks. Restrictive [3], [12], [18]: these define rules or techniques to limit the level of freedom in writing requirements in NL. They are able to mitigate the effects of the inherent ambiguity of the NL, but they address more the needs of requirements engineers rather than users because they make the requirements more precise and analysable but less comprehensible.

Inductive [11], [14], [17]: these identify common problems in NL requirements and propose corrective actions or writing styles. They recommend safe writing style for requirements but do not provide the means to apply them, hence they have a limited impact in practice.

Analytic [2], [13], [20], [21], [25]: these consider NL requirements as they are and, by means of linguistic techniques, an analysis is performed to identify and remove defects. They have a high potential and direct impact in practice, but they have not been studied or developed in depth so are consequently not very precise or effective. The aim of our work is to further develop Analytic methods in order to provide an effective and positive impact in the analysis of NL requirements.

3. Background: A Method for Requirements Analysis

Before describing the QuARS tool (Sections 4 and 5), we will outline the methodology defined in [8, 9, 10] to deal with linguistic problem in NL requirements documents. The first step was to define a quality model against which NL requirements can be evaluated.

The main quality properties that can be addressed and evaluated by means of NL understanding techniques can be grouped into three categories:

- Expressiveness: i.e. the incorrect understanding of the meaning of the requirements, specifically ambiguities and poor readability.
- Consistency: i.e. the presence of semantic contradictions in the NL requirements documents.
- Completeness: i.e. the lack of necessary information.

Lexical, syntactical or semantic analysis of NL requirements can be performed by applying linguistic techniques. We can thus talk about, for example, lexical non-ambiguity or semantic non-ambiguity rather than non-ambiguity in general. For instance, an NL sentence may be syntactically non-ambiguous (in the sense that only one derivation tree exists according to the applicable syntactic rules) but it may be lexically ambiguous because it contains wordings that do not have a unique meaning.

Figure 1 shows that the quality of NL requirements can be represented as a two-dimensional space, where the horizontal dimension consists of the three target properties to achieve (Expressiveness, Consistency and Completeness) and the vertical dimension the various points of view from which the target properties can be considered when linguistic techniques are used.

<table>
<thead>
<tr>
<th>Expressiveness</th>
<th>Ambiguity mitigation</th>
<th>lexical</th>
<th>syntactical</th>
<th>semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Understandability improvement</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consistency</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Completeness</th>
</tr>
</thead>
</table>

Figure 1: Two-Dimensional Representation of the quality of NL requirements

Linguistic techniques are effective in addressing the issues related to Expressiveness because the lexical and syntactical levels provide enough means to detect a large number of defects. A Quality Model is the formalization of the definition of the term “quality” to associate with a type of work product. The typical objectives of a quality model are to define, analyze and document a product.
• Quality Characteristics: define and document the relevant quality factors (also known as quality attributes or “ilities”) that are important attributes of work products (e.g. applications, components or documents) or processes that characterize part of their overall quality (e.g. extensibility, operational availability, performance, re-usability, …). Quality sub-characteristics are important components of quality characteristics.
• Quality Indicators: are specific descriptions of something that provides evidence either for or against the existence of a specific quality characteristic or sub-characteristic.
• Quality Metrics: provide numerical values estimating the quality of a work product or process by measuring the degree to which it possesses a specific quality characteristic.

The Quality Model [9] we defined for the Expressiveness property of NL software requirements, is aimed at providing a way to perform a quantitative (i.e. one that allows the collection of metrics), corrective (i.e. one that could be helpful in the detection and correction of the defects) and repeatable (i.e. one that provides the same output against the same input in every domain) evaluation.

Our definition of the Quality Model was driven by some results in NL understanding, by experience in formalizing software requirements and also by an in-depth analysis of real requirements documents provided by industrial partners. Moreover, we took advantage of experience in the field of requirements engineering and software process assessment using the SPICE (ISO/IEC 15504) model [16]. Although not exhaustive, the model is sufficiently specific to include a significant part of lexical and syntax-related issues of requirements documents.

The quality model for expressiveness consists of three quality characteristics that are evaluated by means of indicators. Indicators are linguistic components of the requirements directly detectable and measurable on the requirements document.

The Expressiveness characteristics are:
• Unambiguity: each Requirement has a unique interpretation.
• Specification Completion: each Requirement can uniquely identify its object or subject.
• Understandability: each Requirement can be fully understood when used for developing software and the Requirement Specification Document can fully understood when read by the user.

Indicators, in this case, are syntactic or structural aspects of the requirements specification documents that provide information on the defects related to a particular property of the requirements themselves. Tables 1, 2, 3 describe the Indicators related to each quality characteristic.

<table>
<thead>
<tr>
<th>Unambiguity Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sub-Characteristics</strong></td>
</tr>
<tr>
<td>Vagueness</td>
</tr>
<tr>
<td>Subjectivity</td>
</tr>
<tr>
<td>Optionality</td>
</tr>
<tr>
<td>Implicity</td>
</tr>
<tr>
<td>Weakness</td>
</tr>
</tbody>
</table>

Table 1: Ambiguity Characteristics

<table>
<thead>
<tr>
<th>Specification Completion Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sub-Characteristic</strong></td>
</tr>
<tr>
<td>Under-specification</td>
</tr>
</tbody>
</table>

Table 2: Specification Completion Characteristic
The sentences recognized as defective according to the quality model described in Tables 1, 2, 3 are not defective sentences according to the rules of general English grammar, but are incorrect in terms of the expressiveness defined above.

The Quality Model also includes a Readability Metric to evaluate Understandability. This metric is given by the Coleman-Liau formula: \(\frac{5.89 \times \text{chars/words} - 0.3 \times \text{sentences}/(100 \times \text{words}) - 15.8}\) [6]. The reference value of this formula for an easy-to-read technical document is 10, if this value is greater than 15 the document is difficult to read.

The quality model was derived taking into account its principal purpose: it should be a starting point for the realization of an automatic tool for the analysis of NL requirements. The indicators the quality model consists of are terms and linguistic constructions that characterize a particular defect and can be detected directly by looking at the sentences of a requirements document.

For this reason in Table 4 some notes explain how the Indicators belonging to the quality model can be highlighted by performing a linguistic analysis of the requirements document.

### Table 3: Understandability Characteristics

<table>
<thead>
<tr>
<th>Sub-characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>the sentence has more than one main verb, subject or object</td>
</tr>
</tbody>
</table>

### Table 4: Expressiveness Defect Indicators

<table>
<thead>
<tr>
<th>Sub-characteristic</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vagueness</td>
<td>Vagueness-revealing wordings (e.g. clear, easy, strong, good, bad, useful, significant, adequate, recent, …)</td>
</tr>
<tr>
<td>Subjectivity</td>
<td>Subjectivity-revealing wordings (e.g. similar, similarly, having in mind, take into account, as [adjective] as possible, …)</td>
</tr>
<tr>
<td>Optionality</td>
<td>Optionality-revealing words (e.g. possibly, in case, if possible, if appropriate, if needed, …)</td>
</tr>
<tr>
<td>Implicity</td>
<td>- Subjects or complements expressed by means of: Demonstrative adjectives (this, these, that, those) or Pronouns (it, they…) or Terms whose determiner is expressed by a demonstrative adjective (this, these, that, those) or implicit adjective (e.g. previous, next, following, last…) or prepositions (e.g. above, below…)</td>
</tr>
<tr>
<td>Weakness</td>
<td>Weak verbs (e.g. can, could, may, …)</td>
</tr>
<tr>
<td>Under-specification</td>
<td>Words that need to be instantiated (e.g.: flow instead of data flow, control flow, access instead of write access, remote access, authorized access, testing instead of functional testing, structural testing, unit testing, etc.)</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>Sentences that have multiple subjects or verbs</td>
</tr>
</tbody>
</table>

What characterizes our approach to the analysis of NL requirements is the application of NL understanding techniques (addressing both lexical and syntactical aspects of NL texts) aimed at performing a quantitative and corrective analysis of requirements. In fact, the occurrences of the quality model’s indicators through the requirements document are pointed out and recorded. Then, improvement opportunities are derived along with corrective actions and metrics.

### 4. QuARS: An Automatic Tool for NL Requirements Analysis

QuARS was developed to automatize NL requirements analysis systematically, on the basis of the quality model described in Section 3. The aim was to develop a modular, extensible tool with a user friendly graphical interface.

QuARS performs the Expressiveness analysis by means of a lexical and syntactic analysis of the input file in order to identify those sentences containing defects according to the quality model. When the Expressiveness analysis is performed, the list of defective sentences is displayed by QuARS and a log file is created. The defective sentences can be tracked in the input requirements document and corrected, if necessary. Metrics measuring the defect rate and the readability of the requirements document under analysis are calculated and stored. The available metrics are:

- The Coleman-Liau Formula readability metrics.
- The defect rate (i.e. the number of defective sentences / the total number of sentences).

In addition to the Expressiveness analysis, QuARS also now includes Requirements Clustering or View Derivation. This means that collections of requirements can be handled in order to highlight clusters of them dealing with specific properties.

These clusters are called Views. The derivation of a View from a document uses special sets of terms containing an appropriate corpus that can be put in relation to specific properties. These sets of terms are grouped into V-dictionaries.

Requirements clustering can provide support for the following tasks:

- Consistency analysis: conflicting, redundant or contradictory requirements can be easily detected by focusing on a cluster where all the requirements deal with the same topic.
- Traceability definition: all the requirements have to be traced in one or more test cases in order to complete a test session. Using clusters that consist of all the requirements describing a specific system function of the to be tested can be helpful for traceability purposes.
- Verification of the correct organization of the requirements document: if a requirements document is structured so that homogeneous requirements are grouped into special sections, then the presence of misplaced requirements can be detected by extracting all those requirements that should be included in the appropriate document section.

In the next section we detail how QuARS analyses NL requirements, and outline its main components.

5. QuARS components

The high-level architectural description of the tool is shown in Figure 2.

![Figure 2: QuARS High-Level Architecture](image)

The input file containing the requirements document to be analyzed is given to the syntax parser, which produces a new file containing the parsed version of its sentences. The tool uses a set of Indicator-related dictionaries that may contain terms indicating a defect type according to the quality model and the V-dictionaries. The dictionaries are in simple text format. Once the user has selected the type of analysis, the corresponding dictionary is made available and if necessary tailored. Not all the aspects related to the automatic support of the analysis/evaluation of requirements quality can be addressed in the same way and with the same depth and ease. The indicators are the basic elements to identify defects and collect metrics. Since the indicators are linguistic components, it is necessary to define precise sets of terms and linguistic constructions to be used for the indicator detection.

The detection of the different indicators the quality model is composed of, entails applying different linguistic techniques. In particular, for certain kinds of indicators only a morphological analysis is needed to search for the terms contained in the related dictionary in the sentences of the requirements document. For other indicators the syntactical
structure of the sentences needs to be derived before pointing out a defect. Table 5 shows the kind of analysis necessary for each of the indicators of the sub-characteristics.

<table>
<thead>
<tr>
<th>Sub-characteristic</th>
<th>Morphological analysis</th>
<th>Syntactical analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vagueness</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Subjectivity</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Optionality</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Implicity</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Weakness</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Under-specification</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Multiplicity</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Readability</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Type of analysis for detecting sub-characteristics related defects

The morphological analysis of the requirements document is sufficient to detect Vagueness, Optionality and Subjectivity defects. In fact, such defects consist of the occurrence of special defect-revealing terms in the requirements. To highlight the other indicators, knowledge of the syntactical structure of the sentences is required. In fact, the implicity indicators can be detected if the subjects and the objects of each sentence are known, the weakness indicator needs the identification of the verbs, the under-specification indicator needs to know the relationship between nouns and modifiers, and the multiplicity indicator needs to know what elements of a sentence are the subjects and the verbs.

QuARS consists of the following main logic modules in order to cover the analysis of NL requirements according to the above considerations:

- Syntax Parser
- Dictionaries
- Indicator Detector
- View Deriver
- Graphical User Interface

5.1 Syntax Parser

This component, using a special purpose grammar, builds the derivation tree of each sentence. During the analysis process, the syntactic category is associated with each term. To implement this component some free software, Minipar [15], was used. This application, which is available on the web, associates tags with the terms of the sentence. These tags indicate the syntactical role of each of the terms. The relations between the syntactical components of the sentence are derived too. For example the sentence:

*The system shall provide a user manual*

is syntactically analyzed as shown in Figure 3.

```
> (  
  1  (the ~ Det 2 det )  
  2  (system ~ N 4 s )  
  3  (shall ~ Aux 4 aux )  
  4  (provide ~ V E0 i (gov fin))  
  5  (a ~ Det 7 det )  
  6  (user ~ N 7 nn )  
  7  (manual ~ N 4 obj )  
)
```

Figure 3: Output of the syntax parser

This outcome has to be interpreted as follows:
To derive the syntactical structure of the sentences one of the possible derivation trees is calculated on the basis of the rules of English language. More than one derivation tree may exist for the same sentence, and in this case the parser may provide multiple syntax recognitions of a sentence.

5.2 Dictionaries

Dictionaries are the passive components of the tool. They contain sets of terms that are needed to perform the Expressiveness analysis (Indicator-related dictionaries) and the View derivations (V-dictionaries). The number and the content of these dictionaries may vary according to the application domain. The effectiveness of the analysis performed by QuARS strictly depends on the precision, completeness and adequacy-to-domain of the Dictionaries. For this reason special care has been devoted to the Dictionary management in QuARS. Dictionaries are easily modifiable, tailorable and dynamically extensible.

The Indicator-related dictionaries contain the terms and the syntactical elements that allow defects in the NL requirements, according to the given quality model, to be detected. To evaluate NL requirement with respect to Expressiveness, a dictionary is defined for each sub-characteristic related to ambiguity.

V-dictionaries may be composed of those terms dealing with particular quality attributes or “ilities”, e.g. security, safety, efficiency..., that are critical in the particular application domain of the NL requirements document under evaluation.

5.3 Indicators Detector

The indicators detector receives as input the derivation trees, the Indicator-related dictionaries and the requirements document to be analyzed. This component highlights, on the basis of the outcomes of the syntax and the input requirement document, the occurrences of the indicators in individual sentences and writes them in the corresponding log file. This component was developed using C++.

In accordance with the rules of the Quality Model and with the contents of the dictionaries, the Indicators Detector module evaluates the sentences in the NL requirements document.

5.4 View Deriver

QuARS also provides support for consistency and completeness analysis by means of the View derivation functionality. A View is a subset of the input requirements document, consisting of those sentences that deal with particular quality attributes or other non-functional aspects of the system. The View deriver identifies and collects together those sentences belonging to a given View. To do so, it takes information from the corresponding V-dictionary and from the output of the syntax parser. Finally, it counts and identifies the sentences that belong to a View occurring in each (sub-) section of the requirements document. The availability of Views makes the detection of inconsistencies and incompleteness easier because the reviewer only has to consider smaller sets of sentences where possible defects can be found with much less effort.

5.5 GUI Interface

The QuARS GUI has three main frames. The Input Frame allows users to load, display and edit input file containing the requirements for analysis (the supported file format is plain text). The Dictionary Frame allows the user to select display and edit the dictionary corresponding to the type of analysis of interest. The Output Frame displays the results of the analysis. QuARS highlights defective sentences in the Expressiveness analysis, and, in the case of the View derivation, it highlights the cluster of sentences belonging to a View and the graphical representation of distribution of these sentences in the document.

Figures 4 and 5 show the QuARS GUI in relation to an Expressiveness analysis and View derivation.
6. Using QuARS

QuARS has evolved through case studies from an initial prototype to the current reliable (though not yet fully engineered) and user-friendly version. Some of the case studies [9] were industrial projects in telecommunications, automotive, banking and aerospace. These experiments gave us feedback on the:

- Effectiveness of the tool in finding defects.
Frequency and typology of false positives: false positives are sentences indicated as defective which, after the analysis of the results provided by QuARS, are recognized as correct by the requirement engineer.

Effort required to apply the tool and to tailor the dictionaries for specific application domains.

What we learned was that the number of defects found in the NL requirement document depends more on the experience and skill of the requirements engineer than on the company maturity. In fact, when different engineers of the same company are involved in the writing of requirements the quality level of the documents (measured in terms of defect/lines of text) may be significantly different.

Although there were false positives in all the case studies, the rate of false positives with respect to actual defects was rarely over 10% and to some extent depended on the dictionaries not being sufficiently tailored. This fact has more influence in the under-specification analysis which is the most domain-sensitive type of analysis. In this case, the involvement of a domain expert may be needed to better customize the domain dictionaries.

The effort required to analyse a requirements document is relatively low. The preparation of the input document takes the most time, since documents in, for example, MS Word or HTML have to be transferred into plain text. Furthermore, in order to avoid loss of formatting information it may be necessary to make some adjustments before this transformation.

6.1 Case Study: Insurance Requirements Document

In this section, we present the outcomes of a case study performed using the current version of the tool. We used six software requirements documents taken from projects in the field of insurance. These documents have different sizes in terms of number of requirements varying from 28 to 248 requirements. The documents were analyzed by QuARS after an outsourced requirements analysis. The outcomes of these experiments show that the underlying methodology is not only able to highlight more defects than the analysis made in the outsourced company, but can also throw some light on the writing style of the various NL requirements engineers, thus making them aware of their common mistakes. In fact, QuARS revealed that one of the requirement engineers tended to make the same types of mistakes.

Table 6 summarizes the results of these experiments. For each document (D1, ..., D6) used in the experiment the size (i.e. the number of single requirements it is composed of), the number of defects related to a quality model sub-characteristic (e.g. optionality) detected with QuARS and the defect rate are provided along with the value of the readability index. It took less than one hour to carry out the whole set of analyses, using a PC with a Pentium 4 processor.

We later established with the insurance company that some false positives occurred in the Weakness, Implicity and Under-specification analysis results, but in all three cases they were less than 8% of the total number of defects found for that analysis.

<table>
<thead>
<tr>
<th>Doc.Id.</th>
<th>Size</th>
<th>Optionality</th>
<th>Subjectivity</th>
<th>Vagueness</th>
<th>Weakness</th>
<th>Implicity</th>
<th>Under-specif.</th>
<th>ReadabilityIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. defect</td>
<td>No. defect</td>
<td>No. defect</td>
<td>No. defect</td>
<td>No. defect</td>
<td>No. defect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>rate (%)</td>
<td>rate (%)</td>
<td>rate (%)</td>
<td>rate (%)</td>
<td>rate (%)</td>
<td>rate (%)</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>197</td>
<td>0 %</td>
<td>9 %</td>
<td>2 %</td>
<td>15 %</td>
<td>6 %</td>
<td>0 %</td>
<td>10.30</td>
</tr>
<tr>
<td>D2</td>
<td>206</td>
<td>1 %</td>
<td>0 %</td>
<td>9 %</td>
<td>42 %</td>
<td>38 %</td>
<td>0 %</td>
<td>10.33</td>
</tr>
<tr>
<td>D3</td>
<td>28</td>
<td>0 %</td>
<td>0 %</td>
<td>1 %</td>
<td>1 %</td>
<td>1 %</td>
<td>0 %</td>
<td>12.69</td>
</tr>
<tr>
<td>D4</td>
<td>77</td>
<td>0 %</td>
<td>1 %</td>
<td>1 %</td>
<td>0 %</td>
<td>0 %</td>
<td>0 %</td>
<td>8.00</td>
</tr>
<tr>
<td>D5</td>
<td>188</td>
<td>0 %</td>
<td>0 %</td>
<td>3 %</td>
<td>12 %</td>
<td>21 %</td>
<td>11 %</td>
<td>8.99</td>
</tr>
<tr>
<td>D6</td>
<td>61</td>
<td>0 %</td>
<td>0 %</td>
<td>2 %</td>
<td>4 %</td>
<td>5 %</td>
<td>8 %</td>
<td>10.35</td>
</tr>
<tr>
<td>D7</td>
<td>248</td>
<td>0 %</td>
<td>3 %</td>
<td>2 %</td>
<td>19 %</td>
<td>33 %</td>
<td>13 %</td>
<td>14.16</td>
</tr>
</tbody>
</table>

Table 6. Summary of results of the analysis

We were then interested in verifying whether the changes and corrections made when a new version of the document was released, affected the number of Expressiveness defects contained in it. We used two sets of documents: the first consisted of the first six versions of a document containing 76 requirements, the second consisted of the first two versions of a document containing 504 requirements. The results showed that the defects that were discovered using QuARS on the first version of a requirements document were still present in the subsequent versions. This means that they were not detected prior to the use of QuARS thus confirming the usefulness of our tool.
An interesting application of QuARS is to analyse non-functional requirements in order to group them according to different attributes. Non-functional requirements are those requirements not strictly tied to the functionalities of the system. They may relate to system properties, the "ilities", such as reliability and security, or to constraints on the services offered by the systems, such as timing constraints, constraints on development process, standards, etc..

Requirements documents may not be well organized and this means that the non-functional requirements may get mixed up with the functional ones. In such cases, the identification of the non-functional requirements may require a significant effort and there is a risk that they will not be adequately taken into account. The situation seems to be better when the structure of the requirements documents provides special sections where non-functional requirements are grouped together. Unfortunately, once again there is no guarantee that they will not be misplaced. Another common and even more dangerous situation is due to the presence of “hidden” non-functional requirements: a non-functional requirement is hidden in a functional requirement when the host requirement contains both functional and non-functional aspects.

As an example of hidden non-functional requirements let us consider the following:

The system shall provide some facility for authenticating, in four seconds or less, the identity of a system user

The above requirement contains a functional part (The system shall provide some facility for authenticating - , the identity of a system user) which describes what the system is expected to do and a non-functional one (in four seconds or less) giving a timing constraint.

The presence of misplaced and hidden non-functional requirements may lead to problems in a software project. In fact, the traceability of requirements on the architecture components and test cases can be difficult. There may also be negative effects in the requirements analysis process. In fact, detecting inconsistencies, contradictions or redundancies, due to the presence of hidden non-functional requirements, may be difficult. In the case of a structured document with separate non-functional requirements which are explicitly expressed, performing analyses for completeness and consistency may be even harder because it may result in being mistakenly confident that the whole set of non-functional requirements is under control.

One way to overcome the problems due to the presence of hidden and misplaced non-functional requirements is to extract all the requirements dealing with a non-functional aspect (e.g. quality characteristics, design criteria, timing constraints) of the system they describe and put them in a special cluster.

This has several advantages:

- Human detection of inconsistencies, incompleteness and redundancies is facilitated because a simpler and homogeneous set of requirements can be considered.
- Hidden and misplaced non-functional requirements can be highlighted and the risks due to their presence in the requirements document mitigated.
- Conformance to a correct organization of the requirements document can be verified.

QuARS shows considerable potential in finding misplaced non-functional requirements in a specification document. This can be done as a tailored application of the View derivation functionality. We illustrate its use through a real requirements document with eight sections:

1. Requirements for the Functions and Performance of the System.
2. Business, Organizational and User Requirements.
4. Human-Factors Engineering Requirements.
5. Operations and Maintenance Requirements.
7. Design Constraints and Qualification Requirements.

The total number of requirements contained in the document exceeds 300. In the document there is no separation between functional and non-functional requirements; moreover, there are hidden non-functional requirements. The clustering function of QuARS was used to identify those requirements dealing with security issues. A security V-dictionary containing security-related terms was built starting with a corpus taken from a security glossary composed of 144 terms. Because of the structure of the document, the Safety, Security, and Privacy Requirements section should contain the specification of the functions and constraints related to security to be adopted during the development of the software.

To check that no further security-related constraints/properties were placed in other sections of the document, the Security View was applied. The result produced by the application of the clustering function is a View composed of 26 sentences. The graph showing the occurrences of the security-related requirements in the various sections of the document is shown in Figure 6.
Figure 6. Distribution of the security-related requirements in the sections of the document

Note that the highest concentration of security-related requirements is in Section 3 of the document “Safety, Security, and Privacy Requirements”.

Four other sections contain nine misplaced security-related requirements that might cause problems in terms of consistency with the others because they could be ignored at requirements analysis time and during the subsequent development phases (testing included) of this product.

8. Conclusions and Future Work

Numerous techniques and automatic tools for managing requirements are available, many of which are for defining requirements, providing configuration management, and controlling distribution. However, there is lack of automatic support for the quality analysis of NL requirements.

We have presented here an automatic tool, called QuARS, for NL requirements analysis and validation. QuARS is an innovative tool that enables users to analyse NL requirements automatically. The kind of analysis QuARS performs is limited to syntax-related issues of a NL requirements document. We used QuARS in several case studies and it proved to be an effective support in the NL requirements analysis.

It has been designed to be highly adaptable to different application domains by tailoring its Dictionaries. Furthermore, it runs with almost all kinds of textual requirements documents because it works on simple text files which virtually all commercial text editors can be converted into. Even though the View derivation function of QuARS is still in an early phase, it is promising for handling requirements documents with different purposes.

The above features mean that QuARS can deal with many well-known problems in NL requirements analysis, that had previously not been addressed with a satisfactory level of precision, ease of use and adaptability. Indeed, NL requirements analysis carried out by humans can take hours of inspection per defect identified. QuARS does not aim to replace the work of inspectors in its entirety, but it can significantly improve this process.

The effectiveness of the analysis very much depends on the completeness and accuracy of the dictionaries QuARS uses. In particular, the effectiveness and the completeness of the Views strictly depends on the associated V-dictionary: the more precise and complete the V-Dictionary, the more effective the outcome. We are currently defining techniques and methods to enrich the initial dictionaries with additional terms from the corpus of interest, as well as new linguistic techniques to deal with semantic deficiencies in NL requirements.

Acknowledgements

The case studies presented in this paper have been set up in collaboration with Mario R. Barbacci, Robert W. Ferguson and Dennis Goldenson of the Software Engineering Institute, Pittsburgh PA.

Special thanks to Maurice ter Beek for his very useful suggestions.
9. References