An Automatic Tool for the Analysis of Natural Language Requirements

Giuseppe Lami, Stefania Gnesi, Fabrizio Fabbrini, Mario Fusani, Gianluca Trentanni
C.N.R. - Information Science and Technology Institute "A. Faedo", Pisa, Italy

Abstract

The availability of automatic tools for the quality analysis of Natural Language requirements is recognized as a key factor for achieving software quality. Unfortunately, the state of the art and practice witnesses a lack of tools and techniques for the Natural Language requirements analysis. This paper presents a methodology and a tool (called QuARS - Quality Analyzer for Requirement Specifications), for analyzing Natural Language requirements in a systematic and automatic way.
The QuARS tool allows the requirements engineers to perform an initial parsing of the requirements for automatically detecting potential linguistic defects that can determine ambiguity problems at the following development stages of the software product. This tool is also able to support automaticaaly the consistency and completeness analysis by clustering the requirements according to a specific topic.

1. Introduction

The achievement of the quality of software requirements is the first step toward software quality. It is well known that inaccuracies in requirement documents could determine serious problems at all the consequent phases of software development. The availability of methods and tools for the analysis of software requirements may improve the effectiveness of the requirement process and the quality of the final product.
The problem of the analysis of software requirements with respect to some quality characteristics (e.g. unambiguity, consistency and correctness) has been extensively exploited in recent years. The problem of requirements quality is usually approached in the research community by using a more formalized way to express them. Formal methods are mathematic-based representations of the requirements allowing a precise validation [1, 5, 22]. The advantages that can be obtained by means of these methods have some costs in terms of the necessity of skilled people for the definition and also for the use of them. Moreover, the communication mechanism between all the stakeholders (customer included) may be difficult because not all of them can be able to understand and manage such a requirements formalism. These are the main reasons why formal methods are usually employed only in particular application domains as for example safety-critical applications. For the largest part of the software project the mean for representing requirements is Natural Language (NL).
When a formal method is applied the initial requirements document is always written in NL and from it the formal specification is derived (see Figure 1). Possible defects in terms of ambiguity in the initial NL document can be moved into the formal version of requirements. The passage from the initial informal representation of requirements and the (first) formal representation of them is

the most critical point when formal methods are adopted. Performing an analysis of NL requirements to detect and remove ambiguity defects is of interest also to reduce the gap between the NL representation of requirements and the following representation made with formal methods.

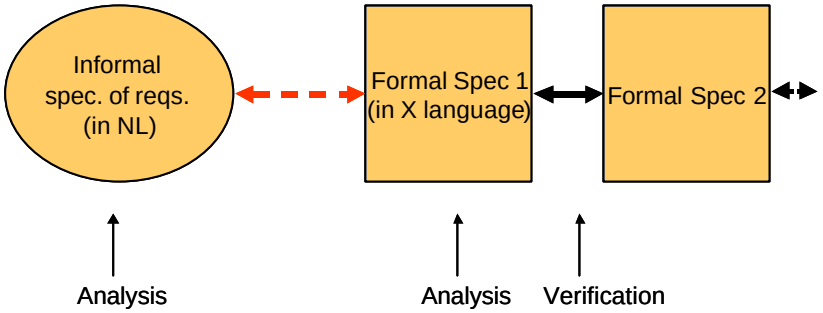

Figure 1: The Requirements Formalization Process Scheme

The use of NL for specifying requirements, even though its inherent ambiguity, indeed has some advantages such as, for example, the ease with which they can be shared among the different people involved in the software development process. In fact, a NL requirement document can be used in different development phases of the final product. For example, it may be used as a working document to be provided as input for system architecture's designers, testers and user manual editors. Moreover, it may be also used to establish an agreement between customers and suppliers and as an information source for the project management.

Unfortunately, the state of the art and practice witnesses a lack of tools and techniques for the NL requirements analysis. The following list, while not exhaustive, includes the most popular practices and means used for mitigating the problems due to the use of NL in requirements specification:

- Tools for analysis: in the literature few descriptions of tools for NL requirements exist. One of the most known tool is Automatic Requirement Measurement (ARM). This tool, developed by NASA, although quite simplistic, is able to perform a lexical analysis for detecting some defects. The defects are mainly identified by means of special terms and wordings that reveal particular defects [23].
- Means for expressing NL requirements: in the practice several means and techniques have been defined in order to mitigate the inherent ambiguity of NL. The most common 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. Another way to express requirements that in recent years is going to be spread in the industry are the Use Case [3]. Use Cases allows a functional description of the requirements and imposes a (light) formalism based on the NL to the requirements.
- Practices for mitigating the effects of the NL inherent ambiguity: because no technique or tool can guarantee the absence of ambiguities in NL requirements, in the practice some countermeasures are frequently adopted. For example, joint reviews of the requirements documents are preformed by customers and suppliers together. The aim of these joint reviews is to verify that the different developers and the customers have the same understanding of each requirement. For doing that the use of glossaries may be of great help.

A recent market research [18] aiming at investigating the existence of potential demand for linguistic tools for requirements analysis concludes: "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 face the demand for NL requirements quality analysis, i.e. the analysis aimed at the detection of ambiguous, incomplete, and contradictory NL requirements. For this reason we defined a method for supporting automatically the analysis of NL requirements in order to extract information from and identify defects in their representation. Such a method, along with the automatic tool that implements it, allows to avoid the tedious, time-consuming and often uneffective manual analysis usually performed by humans.
The paper is structured as follows: in section 2 a survey of the works related to the NL requirements analysis is provided. In section 3 the method we propose for the analysis of NL requirements is shown and in Section 4 the tool that automatize this methodology is described. In Section 5 the experience in the application of the tool to industrial case studies is reported and discussed. Finally, in Section 6 conclusions are provided and future works described.

2. Related works

Several studies dealing with the evaluation and the achievement of quality in NL requirement documents can be found in the literature. We will briefly discuss some of them that we consider to be of particular interest to our research.
Macias and Pulman [17] apply domain-independent Natural Language Processing (NLP) techniques to control the production of natural language requirements.
They propose the application of NLP techniques to requirements documents in order to control:

- 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 to sentences, depending on the degree of syntactic and semantic uncertainty of the sentence. The information is conveyed by discovering under-specifications, missing information, unconnected statements. Finally, they discuss how NLP techniques can help the design of subsets of the English-grammar to limit the generation of ambiguous statements.
Goldin and Berry [12] implemented a tool for the extraction of abstractions from natural language 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.
Hooks [13] discusses a set of quality characteristics necessary to produce well-defined natural language requirements. This paper discusses some common problems which arise when requirements are produced, and looks at how to avoid them. It provides an in depth survey of the principal sources of defects in natural language requirements and the related risks.
Wilson and others [23] examine the quality evaluation of natural language software requirements. Their approach defines a quality model composed of quality attributes and quality indicators, and develops an automatic tool (called ARM: Automated Requirement Measurement) to perform the analysis against the quality model aiming to detect defects and collect metrics.

Fuchs [11] proposes to solve the problems related to the use of NL in requirements documents by defining a limited natural language, called Attempt Controlled English (ACE), able to be easily understood by stakeholders and by any person 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 natural language expressions and to combine these with the rigor of formal specification languages.

Kamsties and Paech [16] focus on the ambiguity evaluation of natural language requirements. They start from the consideration that ambiguity in requirements is not just a linguistic-specific problem and propose the idea of a checklist addressing not only linguistic ambiguity but also the ambiguity related to a particular domain.

Mich and Garigliano [19] 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.

Firesmith [10] provides a, yet not exhaustive, list of the characterisitcs good-quality requirements are expected to exhibit, along with a discussion on the defects requirements are often affected.

Natt och Dag et al. [20] 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 then may 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 the human analysis, aimed at detecting inconsistencies and discrepancies, by focusing on smaller sets of requirements.

The CREWS (Cooperative Requirements Engineering With Scenarios) [4] project aimed at defining an innovative scenario-based method to elicit and validate requirements. The method it proposed, is based on the cooperation beteween users and requirement engineers to define scenarios. Scenarios are considered as a way to express the goal on a format acceptable by both the parties. In particular, in the CREWS-L'Ecritoire defined an approach where the scenarios are expressed in a middle groud 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] propose an approach to the problem of achieving high quality NL requirements that defines a system called CIRCE that can build (semi-) formal models in an almost automatic fashion, extracting information from the NL text of the requirements; the system can then measure and check the consistency of these models. CIRCE can be profitably adopted as a means to induce the use of a suitable style in writing the requirements.

The studies presented in this survey can be classified in three cathegories:

Restrictive: they are based on the definition of rules or techniques for the limitation of the degree of freedom in writing requirements in NL. [17], [11],[4] belong to this cathegory.

Inductive: aiming at identifying common problems in NL requirements and proposing corrective actions or writing styles. [13],[16], [10] belong to this cathegory.

Analytic: they consider the NL requirements as they are and, by meanss of linguistic techniques perform analysis to identify and remove defect. [12], [23], [19],[20], [2].

These different approaches contribute to the improvement of the Requirement Engineer discipline but each of them presents some weaknesses. In particular, the Restrictive approaches are able to

mitigate the effects of the inherent ambiguity of the NL, but they are more oriented to address the needs of the requirement engineers rather than the users because they make the requirements more precise and analysable but less understandable. The Inductive approaches are able to recomment safe writing style for requirements but don't provide means to apply them, then they have a limited impact on the practice. Finally, the Analytic approaches have potentially an high and direct impact on the practice but they have not been deeply studied and developed with the consequence to be scarcely precise and effective.
The objective of our work is to contribute in the development of Analytic methods able to provide an effective and positive impact on the practice.

3. A Method for Requirements analysis (Background)

To achieve quality, NL requirements must embody several properties. In this section those quality properties that can be addressed and evaluated by means of NL understanding techniques are discussed. These properties can be grouped in tree categories:

- Expressiveness: it includes those characteristics dealing with a incorrect understanding of the meaning of the requirements. In particular, the presence of ambiguities in and the inadequate readability of the requirements documents are frequently causes of expressiveness problems.
- Consistency: it includes those characteristics dealing with the presence of semantics contradictions in the NL requirements document.
- Completeness: it includes those characteristics dealing with the lack of necessary information within the requirements document.

The application of linguistic techniques to NL requirements, allows their analysis from a lexical, syntactical or semantic point of view. For this reason it is proper to talk about, for example, lexical non-ambiguity or semantic non-ambiguity rather than non-ambiguity in general. For instance, a NL sentence may be syntactically non-ambiguous (in the sense that only one derivation tree exists according to the syntactic rules applicable) but it may be lexically ambiguous because it contains wordings that have not a unique meaning.
Figure 2 shows schematically that the quality of NL requirements can be represented as a two-dimensional space, where the horizontal dimension is composed of the three target properties to be achieved (Expressiveness, Consistency and Completeness) and the vertical dimension is composed of the different points of view from which the target properties can be considered when linguistic techniques are used.

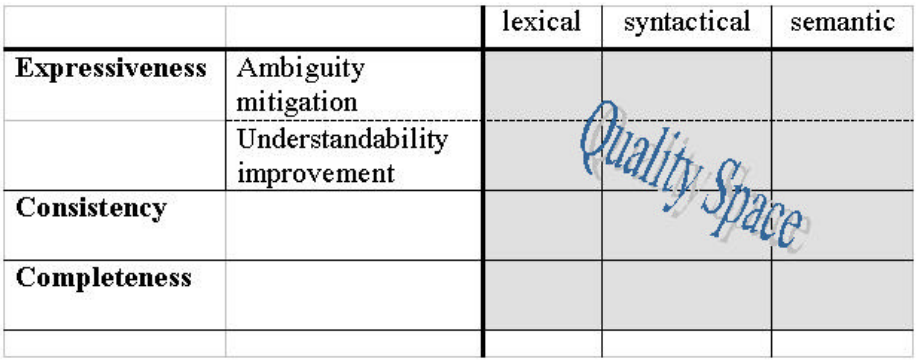

Figure 2: Two-Dimensional Representation of the NL requirements Quality

Linguistic techniques can effectively address the issues related to the Expressiveness because the lexical and syntactical levels provide means enough to obtain effective results.

A Quality Model is the formalization of the definition of the term "quality" to be associated to a type of work product. The typical objectives of a quality model are to define, analyze, and document a product's:

- 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 characterizes part of their overall quality (e.g. extensibility, operational availability, performance, re-usability, …). Quality sub-characteristics are important components of quality characteristics.
- Quality Indicator: 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.

In [8] we have defined a quality model for NL requirements Expressiveness.

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

The definition of the Quality Model has been driven by some results in the natural language understanding discipline, by an experience in formalization of software requirements and also by a depth analysis of real requirements documents provided by industrial partners. Moreover this quality model has been defined after a study of the existing related literature and by taking advantage from matured experience in the field of requirement engineering and software process assessment according to the SPICE (ISO/IEC 15504) model [15]. This quality model, though not exhaustive, is sufficiently specific to include a significant part of lexical and syntax-related issues of requirements documents.

The Expressiveness quality model is composed of three quality characteristics to be evaluated by means of indicators. Indicators are linguistic components of the requirements directly detectable and measurable on the requirement document.
The Expressiveness characteristics are:

- Unambiguity: the capability of each Requirement to have a unique interpretation.
- Understandability: the capability of each Requirement to be fully understood when used for developing software and the capability of the Requirement Specification Document to be fully understood when read by the user.
- Specification Completion: the capability of each Requirement to uniquely identify its object or subject.

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

Unambiguity Characteristic	
Sub-Characterisitcs	Description
Vagueness	the sentence contains items having a non uniquely quantifiable meaning
Subjectivity	the sentence expresses personal opinions or feeling
Optionality	the sentence contains an optional part (i.e. a part that can or cannot considered)
Implicity	the sentence doesn't specifies the subject or object by means of its specific name but uses a pronoun or other indirect references.
Weakness	the sentence contains a "weak" verb. A verb that makes the sentence not imperative is considered weak (i.e. can, could, may, ..)

Table 1: Ambiguity Characteristic

Specification Completion Characteristic	
Sub- Characteristic	Description
Under-specification	the sentence contains a word identifying a class of objects without a modifier specifying an instance of this class

Table 2: Specification Completion Characteristic

Understandability Characteristic	
Sub- Characteristics	Description
Multiplicity	the sentence has more than one main verb, subject or object

Table 3: Understandability Characteristic

The sentences recognized as defective according to the quality model described in Tables 1, 2, 3 are not defective sentences according to the English Language rules. Rather they are incorrect in terms of the above defined expressiveness characteristics.

The Quality Model includes also a Readability Metric to be used for the evaluation of the Understandability characteristic. This metric is the Coleman-Liau Formula readability metrics: *(5.89*chars/wds-0.3*sentences/(100*wds)-15.8])*. The reference value of this formula for an easy-to-read technical document is 10, if it is >15 the document is difficult-to-read.
The quality model has been 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 is composed of are terms and linguistic constructions characterizing a particular defect and being directly detectable 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 pointed out by performing a linguistic analysis of the requirements document:

Sub-characteristic	Indicators
Vagueness	The occurrence of Vagueness-revealing wordings (as for example: clear, easy, strong, good, bad, useful, significant, adequate, recent, ...) is considered a vagueness Indicator
Subjectivity	The occurrence of Subjectivity-revealing wordings (as for example: similar, similarly, having in mind, take into account, as [adjective] as possible, …) is considered a subjectivity Indicator
Optionality	The occurrence of Optionality-revealing words (as for example: possibly, eventually, if case, if possible, if appropriate, if needed, …) is considered a optionality Indicator
Implicity	The occurrence of: - Subject or complements expressed by means of: Demonstrative adjective (this, these, that, those) or Pronouns (it, they…)or - Terms having the determiner expressed by a demonstrative adjective (this, these, that, those) or implicit adjective (as for example previous, next, following, last...) or preposition (as for example above, below...) Is considered an implicity Indicator
Weakness	The occurrence of Weak verbs is considered a weakness Indicator
Under-specification	The occurrence of words needing to be instantiated (for example: 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.) is considered an under-specification Indicator.
Multiplicity	The occurrence of sentences having multiple subject or verb is considered a multiplicity Indicator

Table 4: Expressiveness Defect Indicators

What characterizes our approach to the analysis of NL requirements is the application of NL understanding techniques (both addressing 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 requirement document are pointed-out and recorded. Then, improvement opportunities are derived along with corrective actions and metrics. This approach has been refined in the last years as described in past works. [7, 9].

4. QuARS – An Automatic Tool for NL Requirements Analysis

The tool QuARS has been developed with the aim to making the NL requirements analysis based on the quality model described in Section 3. automatic. To make the methodology described above applicable to real requirements documents, it is necessary to provide an automatic support for detecting the quality model's indicators. The indicators are the basic elements to identify defects and collect metrics. Because of, the indicators are linguistic components, it is necessary to define precise sets of terms and linguistic constructions to be used for the indicator detection. These sets of terms are called Dictionaries. A Dictionary characterizes a quality model sub-characteristic and, for this reason, QuARS performs a sub-characteristic's analysis starting from the corresponding dictionary.

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 the QuARS tool. Dictionaries are easily modifiable, tailorable and dynamically extensible.

The detection of the different indicators the quality model is composed of requires the application of different linguistic techniques. In particular, for certain kinds of indicator the performance of a morphological analysis is sufficient for searching the terms contained in the related dictionary in the sentences of the requirement document. For other indicators it is necessary to derive the syntactical structure of the sentences in the requirements document under analysis before pointing out a defect. In the Table 5 the kind of analysis necessary for each of the sub-characteristics' indicators is provided.

Sub-characteristic	Morphological analysis	Syntactical analysis
Vagueness	X	
Subjectivity	X	
Optionality	X	
Implicity		X
Weakness		X
Underspecification		X
Multiplicity		X
Readability	X	

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

The morphological analysis of the requirement document is sufficient to detect a Vagueness, Optionality and Subjectivity defect, in fact such a defect consists of the occurrence of special defect-revealing terms in the requirements. To point out the other indicators the 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 sentences are known, the weakness indicator needs the identification of the verbs, the underspecification 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.

4.1 Functionality of QuARS

The QuARS (Quality Analyzer of Requirement Specifications) is an automatic tool that support the analysis of requirements documents written in Natural Language.
The QuARS's GUI is composed of three main frames (see figure 3):

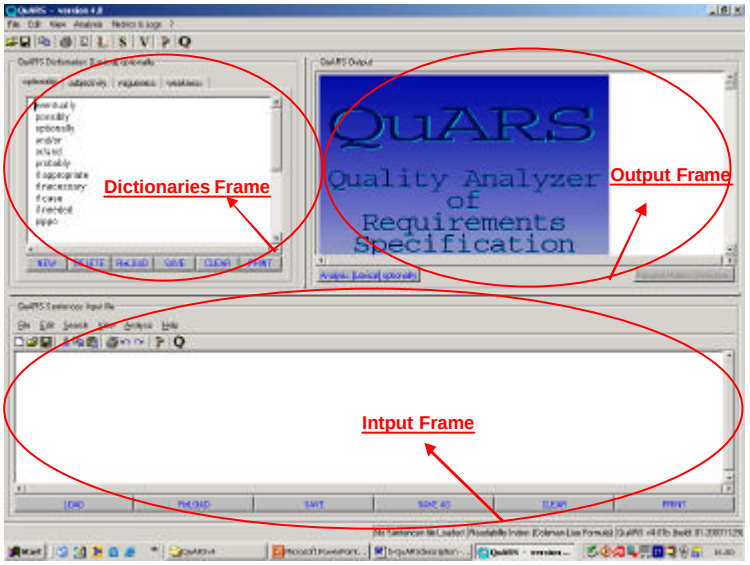

Figure 3: The QuARS GUI

Input Frame: by this frame it is possible to load and display the plain text file containing the requirements to be analyzed. The input frame provides the principal functions (buttons) for editing the input requirements.

Dictionary Frame: this frame allows to select and display and edit the dictionary corresponding to the type of analysis of interest.

Output Frame: in this frame the results of the analysis are displayed.

The functionalities provided by QuARS are:

1. Defect identification: QuARS performs a linguistic analysis of a requirement document in plain text format and points out the sentences that are defective according to the expressiveness quality model described in section 3. The defective sentences can be automatically tracked on the requirement document (in the input frame) to allow their correction.

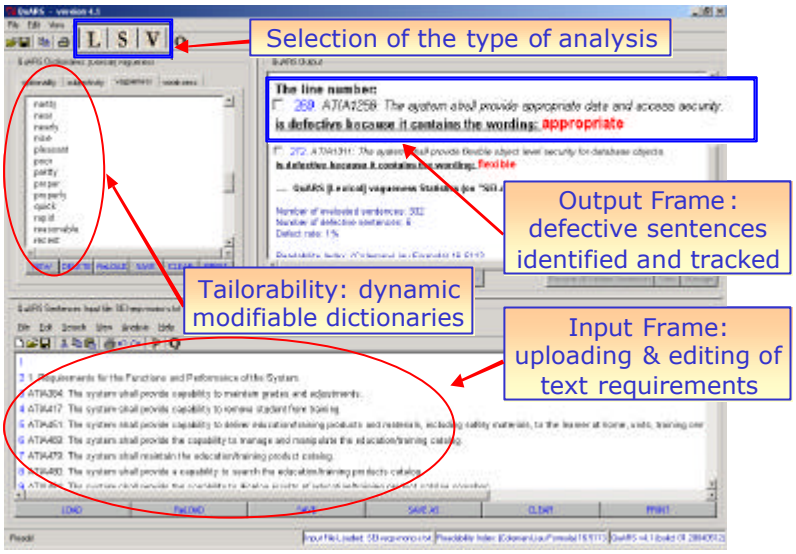

Figure 4: QuARS GUI for the defect identification functionality

2. Requirements clustering: The capability to handle collections of requirements in order to highlight clusters of requirements holding specific properties can facilitate the work of the requirements engineers. In particular, such a functionality can provide support for the following tasks:
   - Consistency analysis: conflicting, redundant or contradictory requirements can be easier detected by focusing on a cluster where all the requirements are dealing with the same topic.
   - Traceability definition: all the requirements have to be traced on one or more test cases in order to make complete a test session. The availability of clusters composed of all the requirements describing a specific function of the system to be tested can be helpful for traceability purpose. In fact, it is the suitability of the test cases defined for testing that function can be verified easily.
   - Verification of the correct organization of the requirement document. If a requirements document is structured so that homogeneous requirements are grouped in special sections, the presence of misplaced requirements can be detected by extracting all those requirements that should be included in the appropriate document section.

   For these reasons, starting from the linguistic techniques developed for the expressiveness analysis, we implemented a functionality that allows to extract clusters of requirements specifying particular properties or capabilities of the system from a textual document and we integrated it in the tool for the expressiveness analysis. These clusters are called Views. The derivation of a View from a document relies on the availability of special sets of terms each of them containing the appropriate corpus that can be put in relation with a particular factor of interest. These sets of terms are called V-dictionaries (or Domain Dictionaries). In the following the defined methodology to derive Views from a NL requirement document is described. First an initial set of terms of V-dictionaries is built by the user on the basis of his/her skill and the study of appropriate technical documentation. In the case study described in section 5, to build an initial V-dictionary for the security View, an security glossary

composed of 144 terms has been used. Then sentences belonging to the same View can be identified automatically by using the output of syntax analyzer and the appropriate V-dictionary. In fact, the sentences having the subject or the object expressed by terms belonging to the V-Dictionary can be tagged as belonging to that View. In other words, the basic idea is to put together those sentences in the requirement document directly or indirectly dealing with a particular topic. The tool provides also the graphical representation of the occurrences of the defects over the different sections the document is composed of.

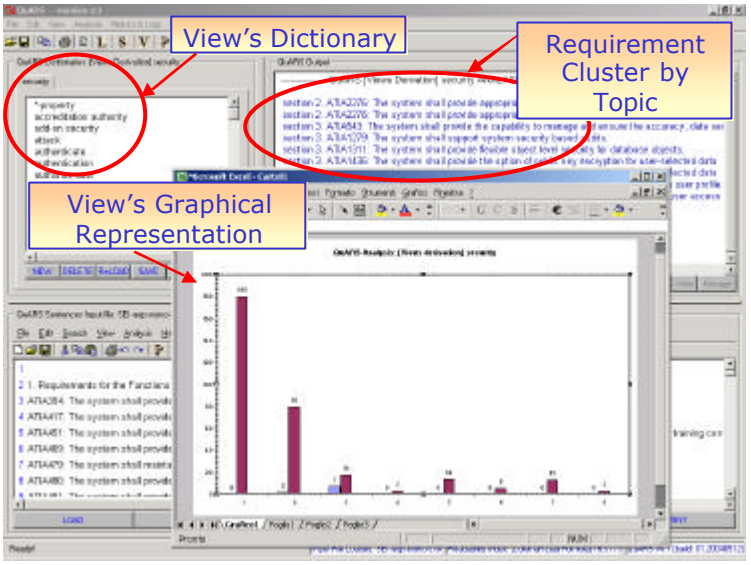

Figure 5: QuARS GUI for the View derivation functionality

3. Metrics derivation: QuARS calculates metrics during the analysis of a requirements document. 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) related to the defects described in 1.

4.2 QuARS Architecture

In this section the high-level architectural description of the tool is provided. The development of the tool has been driven by the objective to be mainly modular, extensible and usable. The architectural design matches the fist two characteristics, the third one has been matched by means of the GUI. In Figure 3 the high level architectural design is depicted.

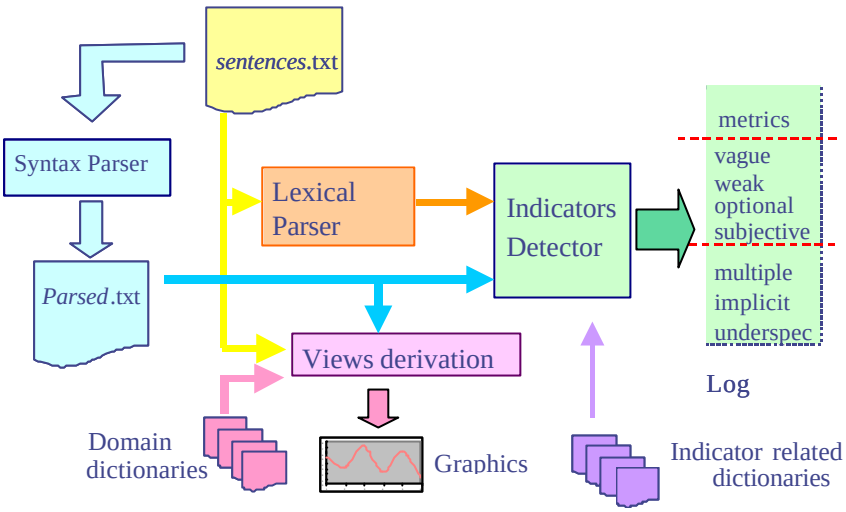

Figure 6: QuARS High-Level Architecture Scheme

The main components of the tool are the lexical and syntax parser, the indicators detector, the view derivator and the dictionaries.

4.3.1 Syntax Parser

This component derives the syntactical structure of each sentence contained in the requirements document. To implement this component the Minipar [14] tool has been used. This application, that is available on the web, associates tags to the terms of the sentence, these tags indicate the syntactical role of each of them. The relations among the syntactical components of the sentence are derived too. As an example the sentence:

1. The system shall provide the manual of the user

Is syntactically analyzed as shown in figure 7.

```
> (
1     (the  ~ Det2     det)
2     (system     ~ N   4     s)
3     (shall      ~ Aux4     aux)
4     (provide    ~ V   E0    i     (gov fin))
5     (the  ~ Det6     det)
6     (manual     ~ N   4     obj)
7     (of   ~ Prep     6     mod)
8     (the  ~ Det9     det)
9     (user~ N   7     pcomp-n)
)
```

Figure 7: output of the syntax parser

This outcome has to be interpreted as follows:

```
line 1: the term the is the determiner (tag Det) of the noun system
at line number 2 (tag 2)
line 2: system is a noun (tag N) and it is the subject (tag s) of
the verb at line 4 (tag 4)
line 3: shall is the auxiliary (tag Aux) of the verb at line 4 (tag
4)
line 4: provide is the main verb (tag V)
line 5: the is the determiner of the noun at line 6 (tag 6)
line 6: manual is a noun (tag N) playing the role of object (tag
obj) of the verb at line 4 (tag 4)
line 7: of is a preposition (tag Prep) of the term at line 6 (tag
6) and it plays the role of modifier of it (tag mod).
line 8: the is the determiner (tag Det) of the term at line 9 (tag
9).
line 9: user is a noun (tag N) playing the role of complement (tag
pcomp-n) due to the term of at line 7 (tag 7)
```

To derive the syntactical structure of the sentences one of the possible derivation tree is calculated on the basis of the rules of English language. It could be possible that more than one derivation tree exists for the same sentence, and in this case the parser may provide a wrong syntax recognition of a sentence. This problem is common to all the existing syntax parser for English sentences (Minipar guarantees an high rate of correctly derived sentences: about 85%).

4.3.2 Lexical Parser

This component is based on the identification of the single terms appearing in the sentences belonging to the requirements document. This component is used to perform a morphological analysis of the sentences.

4.3.3 Indicators Detector

This component points out, on the basis of the outcomes of the syntax and lexical parsers, the occurrences of the indicators in the single sentences of the document under analysis and writes them in the correspondent log file. This component has been developed, using the C++ language.

4.3.4 View Derivator

This component, acquires, as first step, the structure of the document in terms of sections and sub-sections. Then identifies and collects together those sentence belonging to a given View. For doing that it takes information from the corresponding V-dictionary and the output of the syntax parser. Finally, it counts the number of sentences that belong to a View occurring in each (sub-) section of the requirements document.

4.3.5 Dictionaries

Dictionaries are the passive components of the tool. They contain sets of terms that are necessary to perform syntactical, lexical analysis and View derivations. The number and the content of these dictionaries may vary according to the application domain and user needs.

4.3.6 Input and Output

The input of the tool is composed of:

- the requirements document to be analyzed. The allowed format of this input file is plain text format (.txt, .dat, ...);
- the indicator-related dictionaries. They contain defect-revealing terms corresponding to a sub-characteristic of the quality model.
- The V-dictionaries. They contain terms related to both functional and non-functional aspects used to identify the sentences belonging to the View.

The Output of the tool are:

- the log files containing the indications of the sentences containing defects. A log file for each kind of analysis is produced.
- the calculation of metrics about the defect rates of the analyzed document.
- The cluster of sentences representing a View and the graphical representation of number of these sentences over the sections of the analyzed document.

5. Using QuARS

The QuARS tool has evolved from an initial prototype status to the current reliable and user-friendly version (that, nevertheless, can't be considered a fully engineered tool yet) after subsequent case studies aimed at evaluating the effectiveness of the methodology and identifying improvement opportunities both in terms of usability and functionality. In the past case studies [8] we used requirements documents taken from real industrial projects. These projects belonged to different application domains: telecommunications, automotive, banking and aerospace. They, not only provided us with feedbacks to evolve the tool itself, but allow us to have a database containing information on the:

- Effectiveness of the tool in finding defects;
- Types and frequency of false positive in the analysis
- Effort required to apply the tool and to tailor the dictionaires for specific application domains.

From such a database of empirical data some general considerations can be made.

The number of defects found depends more on the experience and skill of the requirements editor than the company maturity. In fact, different requirements document produced by the same Organizational Unit may have a different quality level (measured in terms of defect/lines of text) according to the experience, skill and familiarity with the application domain of the rerquirements editor.

The presence of false positive has been observed in every case study. It can be considered as a physiological side effect of the application of the tool. The rate of false positive respect to actual defects, rarely has been over 10%. The accuracy of the tailoring of the dictionaries according to the

specific application domain has been observed to have a scarse influence on this rate, but in the case of the under_specification analysis that is the most domain-sensitive type of analysis. In this case, the involvment of a domain expert may be needed.
The effort required to perform tha analysis of a requirements document is relatively low. In fact, the main effort is due to the preparation of the input document. Because the input document shall be in plain text format, it is necessary to transform any other possible formats (e.g. MS Word, HTML,..) in text documents. Furthermore, in order to avoid loss of information, it could be necessary to make some adjustments before this transformation. The time QuARS takes to perform the analysis of a requirements document composed of 200 sentences is less than 2 minutes.

Application of the defect identification function

We present the outcomes of a case study performed using the current version of the tool. In this case study we used six software requirements documents taken from projects in the assurance domain. These documents have different sizes in terms of number of single requirements varying from 28 the 248 requirements. These documents were analyzed by QuARS after an outsourced requirements analysis. The outcomes of these experiments shown that the underlying methodology not only can be able to point out additional linguistic defects respect the analysis made by the outsourced company, but can provide also some indications on the writing style of the different NL requirements editors giving them the opportunity to be aware of their common mistakes for avoiding them in the future. In particular, it has been noted that a requirement editor is inclined to repeat the same types of mistakes.

In Table 6 a sketch of the results of these experiments is provided. For each document used in the experiment (in the table indicated as D1, …, D6) the size (i.e. the number of single requirements it is composed of), the number of defects related to a quality model sub-characteristic (in particular, subjectivity, optionality, vagueness, weakness, underspecification and implicity) detected with QuARS and the defect rate are provided along with the value of the readability metric.

Doc. Id.	Size	Optionality		Subjectivity		Vagueness		Weakness		Implicity		Underspecif.		Readability Index
		n.	defect rate	n.	defect rate	n.	defect rate	n.	defect rate	n.	defect rate	n.	defect rate	
D1	197	0	0 %	9	4 %	2	1 %	15	7 %	6	3 %	0	0 %	10.30
D2	206	1	0.5 %	0	0 %	9	4 %	42	20 %	38	18 %	0	0 %	10.33
D3	28	0	0 %	0	0 %	1	3 %	1	3 %	1	3 %	0	0 %	12.69
D4	77	0	0 %	1	1 %	1	1 %	0	0 %	0	0 %	0	0 %	8.00
D5	188	0	0 %	0	0 %	3	1 %	0	0 %	12	6 %	21	11 %	8.99
D6	61	0	0 %	0	0 %	2	3 %	1	1 %	4	6 %	5	8 %	10.35
D7	248	0	0 %	3	1 %	2	1 %	3	1 %	19	7 %	33	13 %	14.16

Table 6. Synthetic results of the analysis of requirements documents with QuARS

The strict collaboration estabished with the industrial partner in conducting this case study, allowed the identification of the false positives. Some false positives occurred in the Weakness, Implicity and Underspecification analysies results, but in all the three cases they were less that 8% of the total number of defects found for that analysis. Other intersting empirical evidences have been gathered by analyzing subsequent versions of a requirements document. The objective of this further experiment was to investigate on possible variations of the defect rate over the different versions of a requirements document during the project. In other words, we was interested in verifying if the changes and corrections made when a new version of the document is released affect the number of Expressiveness defects contained in it. We used two sets of documents: the first was composed of the first six versions of a document containing 76 requirements, the second was composed of the first two versions of a document containing 504 requirements. The evidences collected shown that the defects that were inserted in the first version of a requirements document were never found prior to the use of QuARS. This observation enforced out idea that the tool is able to integrate the human Expressiveness analysis of a requirement document.

Application of the View derivation function to cluster Non-Functional requirements

In practice, requirement documents may suffer from a lack of organization: in particular, the non-functional requirements may not be explicitly differentiated from the functional ones. This leads to the presence of non-functional requirements "misplaced" among the functional ones in the requirement document. In such a case, the identification of the non-functional requirements may require a significant effort and the risk of not taking them into account appropriately exists. The situation seems to be better when the structure of the requirement documents provides for special sections where non-functional requirements are grouped together. Unfortunately, also in this situation there is no guarantee that no misplaced non-functional requirements exist anyway. 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 requirements when the host requirement contains both functional and non-functional aspects.

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

An extensive online help facility, having a tree structure, shall be provided by the system.

The requirements above contains a functional part (*An extensive online help facility - shall be provided by the system*) that describes what the system is expected to do and a non functional one (*having a tree structure*) describing how it has to be done.

The presence of misplaced and hidden non-functional requirements may determine problems in a software project. In fact, the traceability of requirements on the architecture components and test cases can be difficult. Negative effects can be induced also in the requirements analysis process. In fact, the analysis aiming at detecting inconsistencies, contradictions or redundancies, because of the presence of hidden non-functional requirements, may be difficult. In the case of a structured document with non-functional requirements separated and explicitly expressed, performing analyses for completeness and consistency can be even harder because it may result in a misleading confidence in having the whole set of non-functional requirements under control.

A 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

factor, design criteria, implementation constraints, …) of the system they describe and put them in a special cluster.
In such a way, multiple advantages can be obtained:

- The human analysis for detecting inconsistencies, incompletenesses and redoundancies is facilitated because a simpler and homogeneous set of requirements can be considered.
- Hidden and misplaced non-functional requirements can be put in evidence and the risks due to their presence in the requirement document mitigated.
- The conformance to a correct organization of the requirement document can be verified.

QuARS described above shows considerable potentialities to face these problems: in this section we will focus our attention on its capability to find misplaced non-functional requirements in a specification document.
In the following the application of the View derivation functionality to a real requirements document is described.
The document used as an example contains requirements specifications for a complex system. It is composed of 8 sections: 1. Requirements for the Functions and Performance of the System, 2. Business, Organizational and User Requirements, 3. Safety, Security, and Privacy Requirements, 4. Human-Factors Engineering Requirements, 5. Operations and Maintenance Requirements, 6. System External Interface Requirements, 7. Design Constraints and Qualification Requirements, 8. Personnel, Training and Logistics 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 hidden non-functional requirements exist (in other words there are requirements containing both functional and non functional issues). The clustering function of the QuARS tool has been applied to identify those requirements dealing with security issues. For this purpose, a special dictionary containing security-related terms has been built starting from a security corpus taken from a security glossary composed of 144 terms. Because of the structure of this document, it can be expected to find in the Safety, Security, and Privacy Requirements section the specification of the functions and constraints - related to security - to be adopted during the development.
In order to verify that no further security-related constraints/properties have been placed in other sections of the document, the Security View has been derived. The result produced by the application of the clustering function is a View composed of 25 sentences. The graph showing the occurrences of the security-related requirements in the different sections of the document is shown in figure 7.
We can notice that in the Section 3. of the document there is the highest concentration of security-related requirements, but they indicate also that in other four sections something about security has been stated. These are nine misplaced security-related requirements that may determine problems in terms of consistency with the others because, due to their misplacement, they could be ignored at requirements analysis time. Furthermore these nine misplaced requirements could specify some constraints that, could be missed during the development phases (testing included) of this product if not well emphasize in the requirement document.

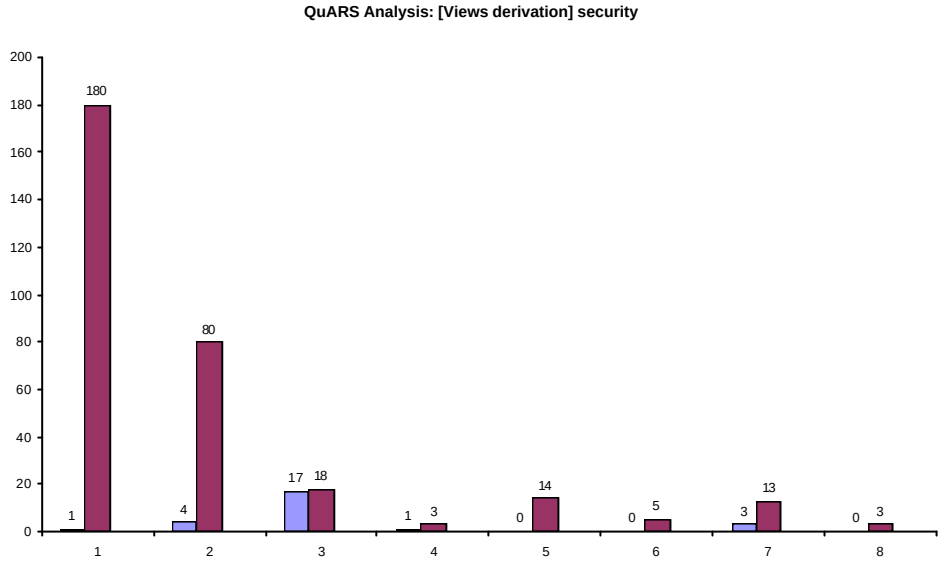

Figure 7. Distribution of the security-related requirements over the document's sections

Metrics Derivation

The availability of metrics allows a synthetic evaluation of some characteristics of the requirements document as for example the writing style or the conciseness. In [6] we adopted a previous version of the QuARS tool (already having the principal functionalities related to he expressiveness analysis implemented) for deriving metrics. Some considerations on how interpret metrics values are provided there.

6. Conclusions and Future Works

In practice, there are numerous tools and techniques for managing requirements. Many are oriented to define requirements, provide configuration management, and control distribution. However, there is a scarcity of automatic support for the quality analysis of Natural Language requirements.

In this paper an automatic tool, called QuARS, for NL requirements analysis and validation has been presented. In the practice, in the software industry the analysis of software requirements is made by humans with a clerical and tedious process that consists of reading of requirements documents looking for linguistic defects. QuARS is an innovative tool that provides the user with the capability of performing the analysis of NL requirements in an automatic way. The kind of analysis QuARS performs is limited to syntax-related issues of a NL requirements document. Only linguistic defects are addressed, while morphological and semantic-related problems are not directly addressed by QuARS. Nevertheless, the application of the QuARS tool to several case studies demonstrates that it can provide an effective support for the NL requirements analysis. In particular, it has been designed to be highly adaptable to different application domains the requirements may belong to. This adaptability can be obtained by tailoring the Dictionaries of the tool to the context of use. Furthermore, it is able to run with almost all kind of textual requirements

because it ask simple text files as input, and it is possible to move to a text file from almost all the formats produced by commercial text editors. The View derivation function of QuARS, even though quite coarse grained yet, is promising can be adopted for handling requirements documents with different purposes.

The effectiveness of the analysis preformed by the tool strongly depends on the completeness and accuracy of the dictionaries it uses. In particular, the effectiveness and the completeness of the Views strictly depends on the associated V-dictionary: the more the V-Dictionary is precise and complete, the more the effectiveness of the outcomes increases. Depending on the technique used and on the content of the V-dictionary, it is possible that the V-Dictionary misses some relevant terms. It could be possible that some terms generally relating to a View have been omitted or that the document under analysis contains specific terms that only in its context can be considered relating to that View. To face this problem the definition of techniques and methods for enriching the initial dictionaries with additional terms always belonging to the corpus of interest is currently under study. These techniques and methods should be able to include additional terms in the dictionaries from the (syntactical) analysis of appropriate technical documentation.

These features make QuARS a tool albe to face well known problems in NL requirements analysis never addressed with a similar level of precision, ease of use and adaptability. The NL requirements analysis, that in practice is made by humans, can take hours of inspection effort per defect identified. QuARS doesn't aims at replace the work of the inspectors at all, but it is able to improve this process because its ability to identify specific cathegories of defects with high performances.

7. References

[1] Abrial JR. The B Book - Assigning Programs to Meanings. Cambridge University Press, August 1996.

[2] Ambriola V, Gervasi V. Processing Natural Language Requirements, 12th IEEE Conf. On Automated Software Engineering (ASE'97), IEEE Computer Society Press, Nov. 1997.

[3] Cockburn A. Writing Effective Use Cases Addison Wesley, 2001.

[4] Ben Achour C., Souveyet C., Tawbi M. Bridging the Gap between Users and Requirements Engineering: the Scenario-Based Approach, International Journal of Computer Systems Science & Engineering, special issue : Object-Oriented Information Systems, Vol 5 N°1, 1999.

[5] Bolognesi T, Brinksma E. Introduction to the ISO Specification Language LOTOS. Computer Networks, 14 (1), 25-59, 1987.

[6] Fantechi A., Gnesi S., Lami G., Maccari A. Application of Linguistic Techniques for Use Case Analysis. Requirements Engineering Journal, Vol. 8, Issue 3, pages 161-170, Springer-Verlag, August 2003

[7] Fantechi A, Gnesi S, Ristori G, Carenini M, Vanocchi M, Moreschini P. Assisting requirement formalization by means of natural language translation, Formal Methods in System Design, vol 4, n.3, pp. 243-263, Kluwer Academic Publishers, 1994.

[8] Fabbrini F, Fusani M, Gnesi S, Lami G. An Automatic Quality Evaluation for Natural Language Requirements, 7th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'01, Interlaken, Switzerland, June 2001.

[9] Fabbrini F, Fusani M, Gnesi S, Lami G. The Linguistic Approach to the Natural Language Requirements Quality: Benefits of the use of an Automatic Tool, 26th Annual IEEE Computer Society - NASA Goddard Space Flight Center Software Engineering Workshop, Greenbelt, MA, USA, November 2001.

[10] Firesmith D. "Specifying Good Requirements", Journal of Object Technology, Vol. 2, No. 4, July-August 2003. ETH Zurich.

[11] Fuchs NE, Schwitter R. Specifying Logic Programs in Controlled Natural Language, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 1995.

[12] Goldin L, Berry DM. Abstfinder, a prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology, and Evaluation. First International Conference on Requirements Engineering, 1994.

[13] Hooks I. Writing Good Requirements, Proc. Of the Fourth International Symposium of the NCOSE , Vol. 2., 1994.

[14] http://www.cs.umanitoba.ca/~lindek/minipar.htm

[15] ISO/IEC TR 15504 (Parts 1-9), 1998

[16] Kamsties E, Peach B. Taming Ambiguity in Natural Language Requirements, ICSSEA 2000, Paris, December 2000.

[17] Macias B, Pulman SG. Natural Language processing for requirement specifications. In Redmill and Anderson, Safety Critical Systems, Chapman and Hall, 1993.

[18] Mich L., Franch M., Novi Inverardi P. Market research for requirements analysis using linguistic tools, Requirements Engineering Journal Vol. 9, Num. 1, pages 40 – 56, Springer-Verlag, February 2004.

[19] Mich L, Garigliano R. Ambiguity Measures in Requirement Engineering. Int. Conf. On Software Theory and Practice - ICS 2000, Beijing, China, Aug. 2000.

[20] Natt och Dag J, Regnell B, Carlshamre P, Andersson M, Karlsson J. Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven development Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, June 2001.

[21] Sommerville I., Sawyer P. Requirements Engineering A good practice guide. Wiley Ed., 1997.

[22] Spivey JM. The Z Notation: A Reference Manual, 2nd edn., London Prentice-Hall, 1992.

[23] Wilson W.M., Rosenberg L.H. Hyatt L.E. Automated Analysis of Requirement Specifications" Nineteenth International Conference on Software Engineering (ICSE-97), Boston, MA, May 1997.

[24] Zowghi D, Gervasi V, McRae A. Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements, Proc. of the 8th Asia-Pacific Software Engineering Conference, December 2001.