An Automatic Quality Evaluation for Natural Language Requirements

ISTEL-INF.
BIBLIOTECA
Posiz. Archivio
TR-32 (2001)

F. Fabbrini, M. Fusani, S. Gnesi, G. Lami

Abstract

This paper presents a tool called QuARS (Quality Analyzer for Software Requirements Specifications) for the analysis of natural language software requirements. The definition of QuARS was based on a special Quality Model for software requirements. The Quality Model aims to provide a quantitative, corrective and repeatable evaluation of software requirement documents. To validate the Quality Model several real software requirement documents were analyzed using our tool with some interesting results.

Categories: D.2.1 [Software Engineering] Requirements/Specifications; D.2.8 [Software Engineering] Metrics

1. Introduction

The achievement of the quality of software requirements is the first step towards software quality. The process leading to the quality of requirements starts with the analysis of the requirements expressed in natural language (NL) and continues with their formalization and verification (for example by using formal methods).

Despite its inherent ambiguity and informality that determine a difficult proving for correctness, natural language is largely used in the software industry for specifying software requirements. Besides the inherent problems of NL, there are other problems which derive from current practices in the industrial SW development process. For example the volatility of the requirements during the development process and the variable levels of linguistic quality due to the different sources they come from [16], means that NL requirement specifications is considered as highly risky for software projects.

Anyway, the use of NL for specifying requirements indeed has some advantages such as, for example, the ease with which it can be shared among the different people involved in the software development process. In fact, a NL requirement document can be used in different ways, and in different development phases of the final product. For example, it may be used as a working document to be provided as input for architecture designers, testers and user manual editors or it may be also used as an agreement document between customers and suppliers or as an information source for the project manager [15].

It is well known that the presence of inaccuracies in requirement documents and specifications introduces serious risks to all the consequent phases of software development. In particular, it is important to provide methods and tools for the analysis of the NL requirement documents because they represent the starting point of a software project [8], [10], [11].

Several studies dealing with the evaluation and the achievement of quality in natural language requirement specifications can be found in the literature. We will briefly discuss some of the those that we consider to be of particular interest.

Hooks [5] discusses a set of quality characteristics necessary to produce well-defined natural language requirements. This paper presents 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 [18] 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 to perform the analysis against the quality model. Nevertheless, their work does not address some important issues such as the adaptability of the tool for different domains. Moreover, even though their quality model takes into account both syntactic and structural aspects of the requirement document, they don't consider any semantic aspects of the quality of requirements in their analysis.

Other works investigate how to handle ambiguity in requirements. In particular, Kamsties and Paech [7] focus especially 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 put forward the idea of a checklist addressing not only linguistic ambiguity but also the ambiguity related to the particular domain. They identify the principal deficiencies of the solutions which are usually adopted to avoid ambiguity in natural language requirements:

- Lack of acceptance by intended users and unfeasibility: extensive lists of rules are usually ignored because they are hard to use in practice .
- Lack of specificity: current inspection techniques rely on checklist-based reading that are composed of unspecific rules . Usually checklists are also ambiguous.
- Uni-dimensionality: only linguistic ambiguities are addressed while ambiguity with respect to other requirements, the application system domains are not addressed.

Mich and Garigliano [13] put forward a set of measures for semantic and syntactic 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. . Natt och Dag et alt. [14] 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, to automatically determine clusters of requirements dealing with the same arguments may enable human analysis aimed at detecting inconsistencies and discrepancies by focusing on smaller sets of requirements.

Our objective is to present an automatic tool for the analysis of NL requirement documents in order to make detection easier and so that defects can be removed which could cause errors in subsequent development phases.
The first step of our approach was to define a Quality Model against which requirements could be checked in order to remove detectable ambiguities and inconsistencies and incompletenesses from a linguistic point of view. The Quality Model is composed of quality properties to be evaluated by means of quality indicators.
Then an automatic analysis tool, called QuARS (Quality Analyzer for Requirement Specifications), was devised taking into account the Quality Model and was used to evaluate real requirement documents used by industry.

This paper is organized as follows: in Section 2. our Quality Model and methodology for NL software requirements evaluation is described. In Section 3 the functionalities and the principal characteristics of the QuARS tool are described. In Section 4. we discuss the real use of QuARS in the requirement process and we present the results of some case studies. Finally, in section 5, we present the conclusions.

2. Natural Language Software Requirement Specifications (NLSRS) Quality Evaluation

Not all aspects related to the automatic support of the analysis/evaluation of the quality of requirements can be addressed in the same way, or with the same thoroughness and ease. In fact, correctness evaluation of NL requirements, i.e. the verification that the system being constructed is correctly described, needs to be supported by more rigorous methods [12] as for example formal methods [19].
Fortunately, some issues related to the linguistic aspects of NL requirements can also be addressed without increasing the formalism level.
These issues may be grouped into two principal families:

- Expressiveness
- Consistency

The Expressiveness family includes those quality characteristics which deal with theunderstanding of the meaning of the requirements by humans. The following topics should be considered as part of the Expressiveness family:

- Ambiguity mitigation: detection and correction of linguistic ambiguities in the sentences;

- Understandability improvement: evaluation of the understandability level of a requirement document and an indication of the areas which need to be improved;
- Specification completion: detection and correction of those sentences containing parts which need to be better specified or need a more precise formulation.

The Consistency family includes those quality characteristics which deal with the presence of semantic contradictions and structural incongruities in the NL requirement document. The following topics should be considered when looking at the Consistency family:

- Mutual referencing checking: detection of missing or wrong references in the requirement document
- Clustering for discrepancy detection: grouping requirements on the basis of similar arguments in order to facilitate the manual analysis of discrepancies, conflicts and duplications.

These families include quality issues that can be successfully addressed and supported with automatic tools which manage the requirements as they are, i.e. requirements expressed in NL.

2.1 A Quality Model for Natural Language Requirements Specifications

As with any other evaluation process, the quality evaluation of NLSRS has to be conducted against a model. The Quality Model we defined for the NLSRS is aimed at providing a way to perform a quantitative (i.e. that allows collections of metrics), corrective (i.e. that could be helpful in the detection and correction of defects) and repeatable (i.e. that provides the same output against the same input in every domain) evaluation.

The Quality Model was defined starting with the identification of a set of representative and meaningful high level properties which needed evaluating. Then a set of quality indicators which were directly detectable and measurable in the requirement document was defined.

The high level properties of the Quality Model are:

- *Testability*: the capability of each requirement to be assessed in a pass/fail or quantitative manner.
- *Specificity*: the capability of each requirement to provide a precise and specific description of its arguments
- *Understandability*: both the capability of each requirement to be fully understood when used for developing software and of the requirement specification document to be fully understood when read by the user.
- *Consistency*: the capability of the requirements to avoid potential or actual discrepancies.

The above properties don't cover all the possible quality aspects of software requirements which are manageable by a linguistic approach, but they are sufficiently specific to be applied (with the support of an automatic tool) to compare and verify NLSRS documents. Furthermore, on the basis of the existing literature and our experience in the Requirements Engineering and Software Process Assessment, we consider this set of properties as sufficient to include most of the syntax-related issues of a requirement document along with a number of interesting structural issues which affect the semantics of the sentences.

The above properties can be automatically evaluated by means of quality indicators. Indicators are syntactic or structural aspects of the requirement specification documents related to particular properties of the requirements themselves.

The Indicators can be detected during the parsing of the requirement document. The detection of an Indicator in the requirement document points to a potential defect addressing the correspondent Property so that corrective actions may be easily done.

In order to do this both the single sentences of the requirement document and the whole requirement document structure are analysed. In fact, some quality Indicators investigate the quality of the requirement document, while others investigate the quality of the single component of the document itself (i.e. the requirement sentences).

The following tables describe the Quality Model, in particular:

Table 1 describes the Indicators related to the quality Properties along with examples of the keywords used to detect potential defects in the NLSRS.

Property	Indicator	Notes
TESTABILITY	**Optionality** Indicator: it reveals a requirement sentence containing an optional part (i.e. a part that can or cannot be considered)	Optionality-revealing wordings: *possibly, eventually, if case, if possible, if needed, ..*
	Subjectivity Indicator: it is pointed out if sentence refers to personal opinions or feeling	Subjectivity-revealing wordings: *similar, better, similarly, worse, bearing in mind, take into account, as [adjective] as possible*
	Vagueness Indicator: it is pointed out if the sentence includes words which are inherently vague, i.e. words having a non uniquely quantifiable meaning	Vagueness-revealing wordings: *clear, easy, strong, good, bad, efficient, useful, adequate, fast, ...*
	Weakness Indicator: it is pointed out when a sentence contains a weak main verb	Weak verbs: *can, could, may.*
SPECIFICITY	**Underspecification** Indicator: it is pointed out when the subject of the sentence identifies a class of objects and doesn't contain a modifier specifying an instance of that class.	
CONSISTENCY	**Undereference** Indicator: it is pointed out in a NLSRS document when a sentence contains explicit references to: • not numbered sentences within the NLSRS document itself • documents not referenced within the NLSRS document itself • entities not defined nor described within the NLSRS document itself	-
UNDERSTANDABILITY	**Implicity** Indicator: it is pointed out in a sentence when the subject is generic rather than specific.	Subject expressed by: demonstrative adjectives (*this, these, that, those*) or pronouns (*it, they, ..*). Subject specified by: adjectives (*previous, next, following, last,..)* or prepositions (*above, below, ..*).
	Multiplicity Indicator: it is pointed out in a sentence if the sentence has more than one main verb or more than one direct or indirect object that specifies its subject	Multiplicity-revealing words: *and, or, and/or, ...*
	Under explanation Indicator: it is pointed out in a NLSRS document when a sentence contains acronyms not explicitly and completely explained within the NLSRS document itself	-

Table 1: Quality Properties and their Indicators

The Indicator keywords used for detecting Indicators in the requirement document were defined after the analysis of a number of requirement documents taken from industrial projects and on the basis of our experience in Requirement Engineering [1], [2], [3], [9] and software process assessment according to the SPICE model (ISO 15504) [6].

2.2 The Evaluation Methodology

Our approach to the problem of the evaluation and improvement of the quality of requirements is to use the defined quality model (and in particular its Indicators) to underline potential defects without forcing any corrective action. It is then left to the requirement engineer to decide if those sentences that, according the quality model, are considered defective need to be corrected and how. This approach takes into account the information regarding the specific application domain or other environment peculiarities that are part of the expertise of the requirement engineer and are difficult to know a-priori.
The quality Indicators of the Quality Model differ in terms of their scope (see Table 2.): some of them need to analyze single sentences in order to be calculated, others need to analyze the whole requirement document.

Quality Indicator	*Requirement Document Sentences*	*Whole Requirement Document*
Implicity	•	
Multiplicity	•	
Optionality	•	
Subjectivity	•	
Under- specification	•	
Under-reference		•
Under-explanation		•
Vagueness	•	
Weakness	•	

Table 2. Scope of the quality Indicators

Due to the different scope of the Indicators belonging to our quality model, the evaluation methodology we propose is based on a twofold analysis: both the requirement document considered as a whole and the single sentences composing it are evaluated (see Figure 1.). The final quality evaluation and the consequent possible corrective actions concern the defects related to the document structure and those related to single sentences.

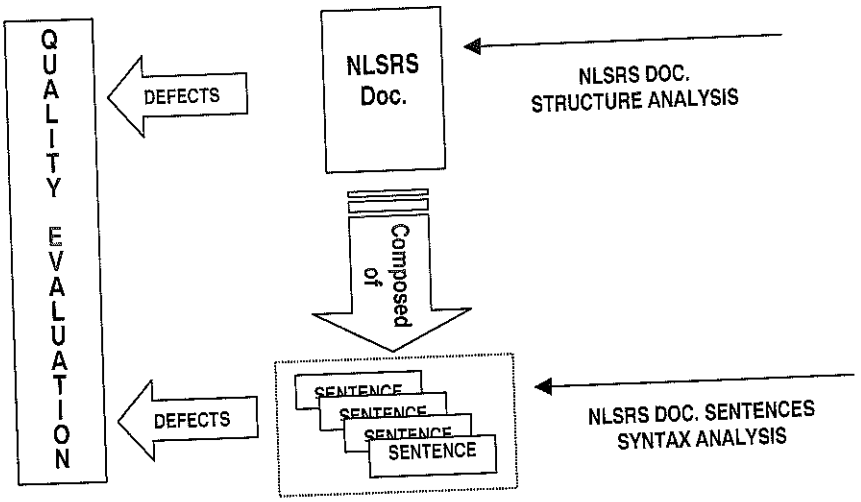

Figure 1. The quality evaluation of a requirement document

3. QuARS: Quality Analyzer for Software Requirement Specifications

In order to make the analysis of natural language software requirements automatic, a tool has been implemented at CNR-IEI. The tool, named QuARS (Quality Analyzer for Software Requirement Specifications) was devised to parse requirement sentences written in English and to point out a number of potential errors in the requirements themselves.
QuARS was developed in the framework of a project[1] that aims to transfer technology to small and medium sizedcompanies in order to improve their software development.
In the tool we developed a linguistic analysis engine which defines a basic English grammar (currently about 40 production rules,) and a small dictionary.
The dictionary contains a set of grammatical words (such as determiners, particles, quantifiers, auxiliary verbs, etc.) manually developed by a computational linguist and a set of "semantic" words (nouns, adjectives, adverbs, verbs) automatically generated by means of the morphological analyzer ENGLEX (http://www.sil.org) running on our test corpus. The dictionary also contains the list of words defined by the AECMA-Boeing Simplified English Project (http://www.aecma.org/Publications/SEnglish/senglish.htm).
For each word, the part of speech and morphological information are stored; grammatical words can also have the grammatical functions defined.
The grammar covers the basic structures of main sentences, direct questions, infinitive clauses, noun phrases and verb groups.
To summarize, the QuARS tool is composed of the following main logic modules (see also Figure 3):

- Lexical Analyzer (ENGLEX – http://www.sil.org)
- Syntax Analyzer
- Quality Evaluator
- Special purpose grammar
- Dictionaries

The phases of the NLSRS quality evaluation carried out by the QuARS tool are described below:

- The files containing the NLSRS document are analyzed by the Lexical Analyzer in order to verify if an appropriate English dictionary has been used. The output of the Lexical Analyzer (i.e. the lexical category associated with each word of the sentence) is the input of the Syntactical Analyzer. , It builds the derivation trees of each sentence, using a special purpose grammar. During the syntactic analysis process, each syntactic node is associated with a feature structure which specifies morpho-syntactic data of the node and application-specific information, such as errors with respect to our quality model.
  In figure 2 the derivation tree and the feature structures of the root node for the sentence "the system is running and the application exits" are shown ("error 07", in the feature structure, corresponds to the criterion that multiple sentences should be avoided)
- The set of derived trees is the input of the Quality Evaluator module of the QuARS tool. The Quality Evaluator module also receives the special dictionaries as input. These dictionaries contain the constructions and the syntactical elements that allow the detection of defects in the NLSRS (see for example the Notes column in Table 1). The Quality Evaluator module, according to the rules of the Quality Model and by reading the dictionaries, performs the evaluation of the sentences. Finally, it provides the user with Warning Messages, that are able to point out which sentences of the NLSRS Document have potential defects.

[1] The LINK project (MURST L.488/92), with the partial support of SSSUP S.Anna, Pisa, Italy.

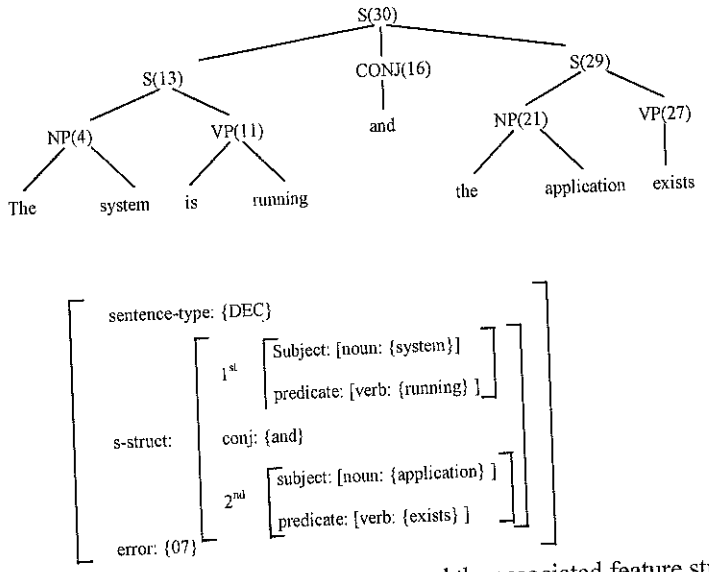

Figure 2: An example of a derivation tree and the associated feature structures

QuARS was designed with the aim of mainly matching the following characteristics:

- Ease of use: the tool has to user friendly both in terms of the people training needed and time consumed.
- Generality: the tool needs to run independently of the format of the NLSRS document.
- Flexibility/tailorability: the tool has to be modifiable in order to make it more effective for particular application domains

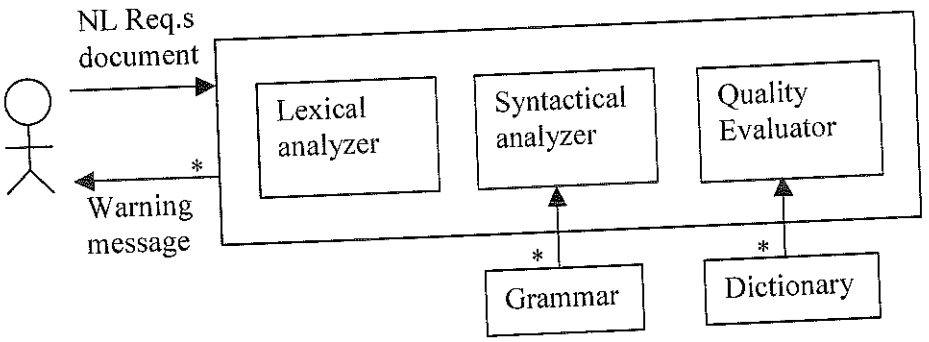

Figure 3 Scheme of the QuARS tool operation

In the following we describe the choices adopted to match these main target characteristics (easy to use, generality and flexibility).

Easy to use: despite QuARS having its own text interface, a TCL/TK [17] graphical interface was released to simplify the use of the tool (see figure 4).

The QuARS GUI (Graphic User Interface) allows the user:

- to carry out an easy and fast navigation in the file system in order to select the file to be analyzed,
- to edit and save the specification file to be analyzed,
- to edit and save all the QuARS dictionaries using the simple but complete built-in editor,
- to easily choose which work session to perform,
- to save the results in a file which can be clearly seen in the wide central window for future use,
- to see the statistics regarding the current work session,
- to provide on line help regarding the purpose of the analysis and the functionality of the tool.

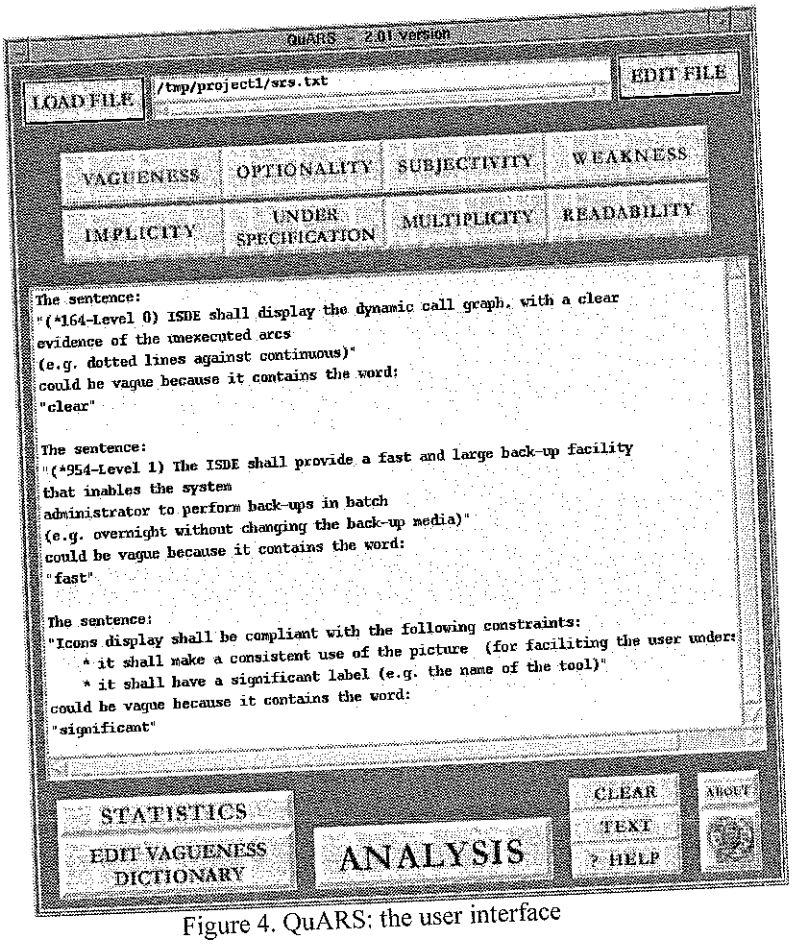

Figure 4. QuARS: the user interface

Generality: *t*he expected format of the requirement document to be evaluated is the text format. This format allows you to achieve high generality because it is always possible to save every format as text format. The risk associated with this strategy is the possibility of leaving out some information related to the particular layout or formatting of the NLSRS document. For example, if multilevel indented bullet lists are used, after the transformation in text format, the hierarchy represented by means of the different levels of indentation will be lost. On the other hand, it can be assumed that in each requirement document the hierarchy of the requirements isn't established by means of the formatting of the text, but has to be defined by means of the appropriate numeration of the sentences. Then the possible lack of information doesn't compromise the validity of the textual requirement document.
Flexibility/Tailorability: *t*he flexibility/tailorability target characteristic is very important for the use of QuARS in industrial domains. In fact, it is important to adapt QuARS to different projects and different application domains. The way QuARS does it is by allowing the user to evolve and modify the dictionaries. Dictionaries are directly used for the detection of several Indicators, and in some cases the content of the dictionaries is strictly dependent on the application domain of the requirement document under evaluation. Some Indicators are more dictionary-sensitive than others (e.g. Vagueness and Under-specification)
Table 3 shows some examples of requirement sentences containing defects detectable using QuARS.

Indicators	*Negative Examples*
Implicity	the above requirements shall be verified by test
Optionality	the system shall be such that the mission can be pursued, possibly without performance degradation
Subjectivity	in the largest extent as possible, the system shall be constituted by commercially available software products
Vagueness	the C code shall be clearly commented
Weakness	the results of the initialization checks may be reported in a special file
Underspecification	the system_ shall be able to run also in case of attack
Multiplicity	the mean time needed to remove a faulty board and restore service shall be less than 30 min.
Undereference	the software shall be designed according to the rules of the Object Oriented Design
Underexplaination	the handling of any received valid TC packet shall be started in less than 1 CUT

Table 3: Examples of requirement sentences containing defects

4. QuARS in the Requirements Process: some case studies

In this section we discuss how the QuARS tool could be integrated within the software requirement process and in particular the software requirement evaluation process. The NL requirement evaluation procedure using QuARS is composed of the following steps:

- Input a requirement document in text format;
- Select an Indicator;
- Run QuARS;
- Use the output of QuARS (indication of those sentences in the requirements document containing potential defects according to the underlying quality model) in order to decide if the document needs modifying

The above procedure can be described more formally as follows:

Let $\mathcal{D}$ be the requirement document under evaluation.

Let Q be the defined Quality Model.

Let I be $\{ i_k \mid k = 1, .., n\}$ where i_k is the k-th indicator of Q.

Let $\mathcal{P}^k{}_Q(\mathcal{D}) \Rightarrow \mathcal{RPD}^k{}_Q$ the function that, given a requirements document $\mathcal{D}$, returns the set of sentences of $\mathcal{D}$ having potential defects according to any indicator i_k.

Let m_k the number of sentences in $\mathcal{D}$ containing potential defects according to i_k.

Hence, $\mathcal{RPD}^k{}_Q = \{ ds^k{}_j \mid j = 1, .., m_k \}$ where $ds^k{}_j$ is the j-th sentence of $\mathcal{D}$ containing potential defects according to the indicator i_k.

We carried out an analysis of real requirement documents taken from industrial software projects using QuARS, in order to better understand if it provides a real support to the improvement of the quality of NL requirements in an industrial environment.

The requirement documents came from different application domains:

$\mathcal{D}1$. Business Application: Functional Requirements of a Transaction and Customer Service (TACS) Check Cashing module;

$\mathcal{D}2$. Space Software Application: Functional Requirements of a sub-system of a space vehicle;

$\mathcal{D}3$. Telecommunication Application: Requirement Specifications of a project aimed at realizing new generation STM switches;

$\mathcal{D}4$. Security Application: Functional Security Requirements for an Application Level Firewall Protection Profile;

We calculated for each i_k of I the corresponding $\mathcal{RPD}^{ik}{}_Q$. The outcomes of the application of QuARS to the above requirement documents are presented in Table 4. which shows, for each evaluated document, the number of indicator occurrences

$\mathcal{D}1$	69	34	1	15	6	0	125	265
$\mathcal{D}2$	97	17	3	48	88	4	257	444
$\mathcal{D}3$	72	95	3	7	5	8	190	395
$\mathcal{D}4$	22	30	0	24	0	3	52	119
(*)	i_1	i_2	i_3	i_4	i_5	i_6	Total defects	Total requirements

(*) i_1: Multiplicity Ind.; i_2: Vagueness Ind; i_3: Optionality Ind., i_4: Underspecification Ind.; i_5. Implicity Ind.; i_6. Subjectivity Ind.

Table 4. Outcomes of the use of QuARS

The outcomes of the use of QuARS on these four case studies show that the occurrences of the potential defects (i.e. those syntactic and structural aspects of a requirement document that according to our Quality Model are pointed out as defects), are significantly high (around 50% of the total number of requirement sentences). According to the defined approach, the requirement engineers, on the basis of their experience and ability, will be free to correct the document, modifying those sentences that they consider are actually defective. In other words, QuARS doesn't force the requirement engineers to follow a particular standard or style but it supports them pointing out potential weak points which need careful management.
The number of elements of each $\mathcal{RPD}^{\zeta}_{Q}$ may then be used to estimate the percentage of distribution of detected defect types. Figure 6. shows that some particular defects are more frequently detected than others by QuARS. In particular, multiplicity vagueness and under-specification seem to be the types of defects affecting a large part of the requirement sentences of a document.

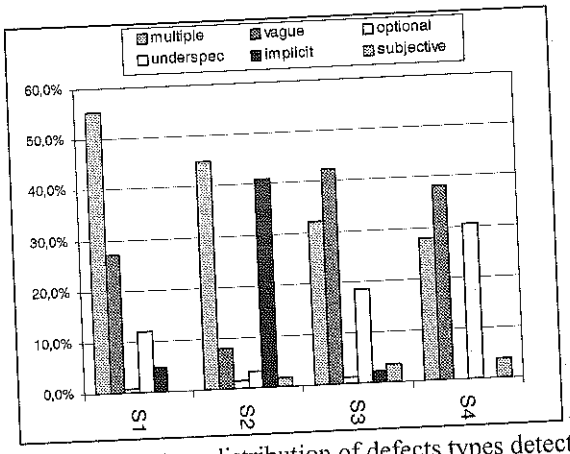

Figure 5. Percentage distribution of defects types detected

Furthermore, a restricted version of QuARS (limited to vagueness, weakness, subjective and optional analysis) was used in a European Space Agency project [4], to analyze the requirements of three real projects (two projects dealing with satellite on-board software and the other dealing with a tool for software production). These documents (containing altogether over 5000 lines of text) were final versions of requirement specifications of ESA projects (then already analyzed, and approved). The results of this analysis are shown in table 4.

Indicators	Defective sentences
Vagueness	5%
Weakness	3%
Subjectivity	3%
Optionality	0%

Table 4. - Results of the use of QuARS in the SPEC project

Previous results show that QuARS was able to identify a significant number defective sentences that were not found by the requirement engineers involved in these ESA projects.

5. Conclusions

In this paper we presented a method for evaluating natural language software requirements according to a previously defined Quality Model. The evaluation of requirement documents following our method aims to support a very critical phase of the software process: the passage from informal requirements (written in natural language) to semi-formal/formal models. The proposed Quality Model was defined with the purpose of detecting and pointing out potential syntactic and structural deficiencies that can cause problems when a natural language requirement document is transformed in to a more formal document. The definition of the criteria used in the Quality Model was driven by some research in the field of natural language understanding (in order to detect the syntactical components introducing for example ambiguity), by our experience in the formalization of software requirements and also by an in depth study of real requirement documents provided by industrial partners. After the definition of the Quality Model against which the natural language requirement documents are evaluated, a tool, named QuARS (Quality Analyser for Requirement Specifications), based on the developed model was developed.
In order to establish confidence regarding the effectiveness of our method/tool, several industrial requirement documents written in natural language were evaluated using QuARS. The outcomes were encouraging because the tool found a large number of potential defects and the requirement engineers of the involved industries found the results useful for improving their requirement documents in that they found in the $\mathcal{RPD}^{k}_{Q}$ a significant set of actual defects occurring in the requirement specifications., The work carried out so far can be continued in two main directions: to enlarge and refine the quality model in order to make the support provided more complete and to make it also able (by modifying parts of the lower level of the tool) to analyze other documents produced in the software development process such as for example: check lists, questionnaires and user manuals.

Acknowledgements

Special thanks to Mr. Giancarlo Gennaro (Intecs HRT) for his collaboration and valuable suggestions.

6. References

[1] F.Fabbrini, M.Fusani, S.Gnesi, G.Lami Quality Evaluation of Software Requirement Specifications, Proc of Software & Internet Quality Week 2000 Conference., San Francisco, CA May 31-June 2 2000, Session 8A2.

[2] F.Fabbrini, M.Fusani, V.Gervasi, S.Gnesi, S.Ruggieri. On linguistic quality of Natural Language Requirements. In 4 th International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'98, Pisa, June 1998.

[3] A. Fantechi, M.Fusani, S.Gnesi, G.Ristori. Expressing properties of software requirements through syntactical rules. Technical Report. IEI-CNR, 1997.

[4] G.Gennaro, D.Lagelle, H.Schabe. Software Product Evaluation and Certification, Proc. Of Data Systems in Aerospace Conference, Nice (France), May 28 - June 1st 2001.

[5] I. Hooks, Writing Good Requirements, Proc. Of the Fourth International Symposium of the NCOSE , 1994, Vol. 2., pp. 197-203.

[6] ISO/IEC JTC1/SC7/WG10 TR 15504 Software Process Assessment Parts 1-9.

[7] E.Kamsties, B.Peach Taming Ambiguity in Natural Language Requirements ICSSEA 2000-5.

[8] J. Krogstie, O.I. Lindland, G. Sindre. Towards a Deeper Understanding of Quality in Requirements Engineering. In 7th International CAiSE Conference, vol. 932 of Lecture Notes in Computer Science, pages 82-95, 1995.
[9] G. Lami Towards an Automatic Quality Evaluation of Natural Language Software Specifications. Technical Report. B4-25-11-99. IEI-CNR, 1999.
[10] F.Lehner Quality Control in Software Documentation Based on Measurement of Text Comprehension and Text Comprehensibility. Information Processing & Management, vol; 29, No. 5, pp 551-568, 1993.
[11] B.Macias, S.G. Pulman. Natural Language processing for requirement specifications. In Redmill and Anderson, Safety Critical Systems, pages 57-89. Chapman and Hall, 1993.
[12] B.Meyer. On formalism in specifications. IEEE Software. January 1985, pages 6-26.
[13] L. Mich, R. Garigliano, Ambiguity measures in Requirements Engineering, Proc. International Conference on Software - Theory and Practice - ICS2000, 16th IFIP World Computer Congress, Beijing, China, 21-25 August 2000, Feng Y., Notkin D., Gaudel M., Publishing House of Electronics Industry, Beijing, 2000, pp. 39-48.
[14] J. Natt och Dag, B. Regnell, P. Carlshamre, M. Andersson, J. Karlsson Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven development Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, June 4-5 2001.
[15] N. Power Variety and Quality in Requirements Documentation Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, June 4-5 2001.
[16] B. Regnell, P. Beremark, O. Eklundh A Market-Driven Requirements Engineering Process: Results From an Industrial Process Improvement Programme, Requirements Engineering, 3(2), 1998, pp. 121-129.
[17] B.Welch Practical Programming in Tcl and Tk second edition Prentice Hall 1997.
[18] W.M.Wilson, L.H. Rosenberg, L.E. Hyatt. Automated quality analysis of Natural Language Requirement specifications. PNSQC Conference, October 1996.
[19] J.M. Wing, J. Woodcock, J. Davies (eds.) FM'99 – Formal Methods, vol. I and II LNCS 1708, 1709, Springer.