The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool

F. Fabbrini, M. Fusani, S. Gnesi, G. Lami
Istituto di Elaborazione dell'Informazione del C.N.R. – Pisa, Italy
{fabbrini, fusani, gnesi, glami @iei.pi.cnr.it}

Abstract

Natural Language (NL) requirements are widely used in software industry, at least as the first level of description of a system. Unfortunately they are often prone to errors and this is partially caused by interpretation problems due to the use of NL itself. This paper presents a methodology for the analysis of natural language requirements based on a quality model addressing a relevant part of the interpretation problems that can be approached at linguistic level. To provide an automatic support to this methodology a tool called QuARS (Quality Analyzer of Requirement Specification) has been implemented. The methodology and the underlying quality model have been validated by analyzing with QuARS several industrial software NL requirement documents showing interesting results.

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 [17], 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 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 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 [16].
It is well known that the presence of inaccuracies in requirement documents could introduce serious problems to all the consequent phases of software development. Hence, it is important to provide methods and tools for the analysis of the NL requirement documents [9], [11], [12].
Our objective has been to realize an automatic tool for the analysis of NL requirement documents in order to easily detect linguistic inaccuracies in them and in a such a way that the defects can be removed as soon as possible.
The first step of our approach was to define a Quality Model against which requirements could be checked in order to remove as much as possible ambiguities, 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), has been devised taking into account the Quality Model and has been used to evaluate real requirement documents used by industry.
This paper is organized as follows: in Section 2. some relevand related works are presented. In Section 3 our Quality Model for NL software requirements evaluation is described. In Section 4 the functionalities and the principal characteristics of the QuARS tool are described. In Section 5 we discuss the use of QuARS in the requirement process and we present the results of its application to some case studies. Finally, in section 6, we present the conclusions.

2. Related Works

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

Macias and Pulman [12] apply domain-independent 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,
- 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 conveyed by requirements, by discovering underspecifications, 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 [6] 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 restricted to a strict lexical analysis of the text.
Hooks [7] 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 [19] 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 aiming to detect defects and collect metrics.
Other works investigate how to handle ambiguity in requirements. In particular, Fuchs [4] 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 into the software development process and simple enough to avoid ambiguities allowing domain specialists to express requirements using natural language expressions and to combine these with the rigour of formal specification languages.
Kamsties and Paech [8] 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 a particular domain.
Mich and Garigliano [14] 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. [15] 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 inter-dependencies 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.

3. Automatic NL Requirements Quality Evaluation

Not all the aspects related to the automatic support to the analysis/evaluation of requirements quality can be addressed in the same way and with the same depth and with the same ease. In fact, correctness evaluation of NL requirements, i.e. the verification that the system to be constructed is correctly described by them, needs to be supported by more rigorous methods [13] as for example formal methods [20].
Fortunately, some issues related to linguistic aspects of NL requirements can be addressed also without increasing the formalism level.
These issues may be grouped into two principal families: Expressiveness and Consistency.
The Expressiveness family includes those quality characteristics dealing with the understanding of the meaning of the requirements by humans. The following topics have to 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 indication of the areas of it needing to be improved;
- Specification completion: detection and correction of those sentences containing parts to be better specified or needing a more precise formulation.

The Consistency family includes those quality characteristics dealing with the presence of semantics contradictions and structural incongruities in the NL requirement document. The following topics have to be considered as addressing to the Consistency family:

- Mutual referencing check: detection of missing or wrong references in the requirement document
- Clustering for discrepancy detection: grouping of requirements on the basis of similar arguments in order

to facilitate the manual analysis of discrepancies and conflicts and duplications.

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

3.1 A NL SRS Quality Model

As any other evaluation process, the quality evaluation of natural language software requirements has to be conducted against a Model. The Quality Model we defined for the natural language 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 quality model we defined is composed of high-level quality properties for natural language requirements to be evaluated by means of indicators directly detectable and measurable on the requirement document.

The high level properties of the Quality Model are:

- *Non-Ambiguity*: the capability of each Requirement to have a unique interpretation.
- *Specification Completion*: the capability of each Requirement to uniquely identify its object or subject.
- *Consistency*: the capability of the Requirements to avoid potential or actual discrepancies.
- *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.

Indicators are syntactic or structural aspects of the requirement specification documents that provide information on a particular property of the requirements themselves. Table 1 describes the Indicators related to the Quality Properties along with examples of the keywords to be used for detecting potential defects in the NL requirements.

PROP.	*INDICATOR*	*DESCRIPTION*	*NOTES*
NON-AMBIGUITY	Vagueness	A Vagueness Indicator is pointed out if the sentence includes words holding inherent vagueness, i.e. words having a non uniquely quantifiable meaning	Vagueness-revealing words*: clear, easy, strong, good, bad, useful, significant, adequate, recent, ...*
	Subjectivity	A Subjectivity Indicator is pointed out if sentence refers to personal opinions or feeling	Subjectivity-revealing wordings*: similar, similarly, having in mind, take into account, as [adjective] as possible, ...*
	Optionality	An Optionality Indicator reveals a requirement sentence containing an optional part (i.e. a part that can or cannot considered)	Optionality-revealing words*: possibly, eventually, if case, if possible, if appropriate, if needed, ...*
	Weakness	A Weakness Indicator is pointed out in a sentence when it contains a weak main verb	Weak verbs*: could, maight.*
SPECIFICATION COMPLETION	Under-specification	An Under-specification Indicator is pointed out in a sentence when the subject of the sentence contains a word identifying a class of objects without a specifyer of this class	Words needing to be more specified: flow (data flow, control flow, ..), access (write access, remote access, authorized access, ..), testing (functional testing, structural testing, unit testing, ..), etc.
CONSISTENCY	Under-reference	An Under-reference Indicator is pointed out in a RSD (Requirement Specifications Document) when a sentence contains explicit references to: not numbered sentences of the RSD itself documents not referenced into the RSD itself entities not defined nor described into the RSD itself	-

PROP.	*INDICATOR*	*DESCRIPTION*	*NOTES*
UNDERSTANDABILITY	Multiplicity	A Multiplicity Indicator is pointed out in a sentence if the sentence has more than one main verb or more than one direct or indirect complement that specifies its subject	Multiplicity-revealing words: *and, or, and/or, …*
	Implicity	An Implicity Indicator is pointed out in a sentence when the subject is generic rather than specific.	Subject expressed by means of: Demonstrative adjective (*this, these, that, those*) or Pronouns (*it, they…*) Subject specified by means of: Adjective (*previous, next, following, last...)* or Preposition (*above, below...*);
	Unexplana-tion	An Unexplanation Indicator is pointed out in a RSD (Requirement Specifications Document) when a sentence contain acronyms not explicitly and completely explained within the RSD itself	-

Table 1. Quality Properties and their Indicators

The Indicator keywords used for detecting Indicators in the requirement document have been defined after the analysis of several requirement documents taken from industrial projects and on the basis of our experience in the Requirement Engineering [1], [2], [3], [10].
The indicators can be detected during the parsing of the requirement document. The detection of an indicator in the requirement document points out a potential problem related to the correspondent Property so that corrective actions may be easily done. The quality indicators differ for their scope: some of them need to analyze single sentences for being detected, others need to analyze the whole Requirement Document (see Table 2.).

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

Table 2. Scope of the Quality Indicators

The sentences recognized as defective according to the quality model described in Table 1. are not uncorrected sentences in terms of English Language rules. Rather they are incorrect in terms of the above defined expressiveness and consistency characteristics.
The defined quality model does not cover all the possible quality aspects of software requirements but it is sufficiently specific for being applied (with the support of an automatic tool) for comparing and verifying the quality of requirement documents.
Furthermore, this set of quality indicators is sufficient to include a large part of the syntax-related issues of a requirement document along with a number of interesting structural-related issues.

4. 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 is named QuARS (Quality Analyzer of Requirements Specifications) and has been realized to parse requirement sentences written in English Language and to point out a number of potential sources of errors into the requirements themselves.
QuARS is composed of the following main logic modules:

- Lexical Analyzer
- Syntax Analyzer
- Quality Evaluator
- Special purpose grammar
- Dictionaries

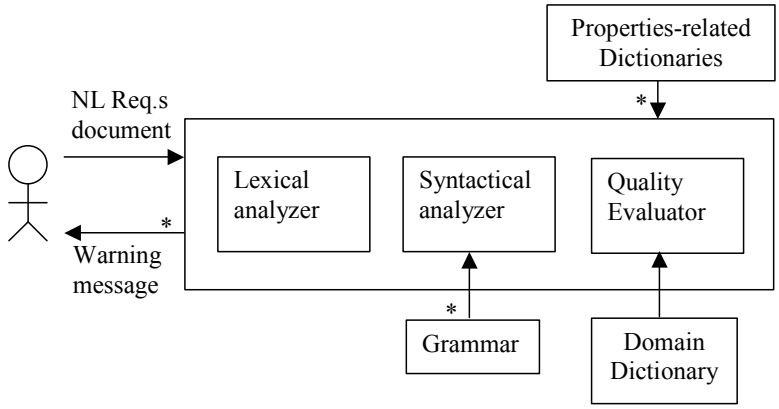

Figure 1. Scheme of the QuARS tool operation

The phases of the SRS Quality Evaluation made by the QuARS tool are described below (see Figure 1.):

- The files containing the SRS Document are analyzed by the lexical Analyzer in order to verify if a correct English Dictionary has been used.
- The output of the Lexical Analyzer (i.e. the lexical category associated to each word of the sentences) is the input of the Syntactical Analyzer. This, using a special purpose grammar, builds the derivation trees of each sentence. During the analysis process, each syntactic node is associated with a feature structure, which specifies morpho-syntactic data of the node and application-specific data, such as errors with respect to our quality criteria.
  In Figure 2 the derivation tree and the features 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 receives also the Properties-related Dictionaries as input. These Dictionaries contain the words and the syntactical elements that allow the detection of inaccuracies in the SRS sentences (see for example the Notes column in the Table 1.). Another Dictionary is used by the Quality Evaluator module: the Domain Dictionary that will contain specific terms of the particular application domain. The Domain Dictionary is used to avoid the detection as defective of terms that belong to the Properties-Related Dictionaries and in the same time they are typical of the application domain. For example, in a financial application domain the wording *clear account* doesn't introduce any vagueness even though the term clear is a vague word. 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 those sentences of the SRS Document having potential defects. Furthermore, some metrics (Readability Index, Comment Frequency and Directives Frequency) are provided too.

QuARS has been designed with the aim to match mainly the following characteristics:

- Ease to use: the tool has to be usable with little effort both in terms of people training needed and time consumed. This characteristic has been matched by means of a TCL/TK [18] graphical interface.
- Generality: the tool shall run independently of the format of the SRS document.
  For this reason the expected format of the requirement document to be evaluated is the text format. This format allows achieving a high generality because it is always possible to save every electronic format as text format.
- Tailorability /flexibility: the tool is modifiable in order to make it more effective for particular application domains.
  The flexibility/tailorability characteristic is very important for the use of QuARS in industrial domain. In fact, it could be necessary to adapt QuARS to different projects and different application domains. The way QuARS does it is by allowing the used to evolve and modify the dictionaries. Dictionaries are directly used for the detection of many indicators, and in some cases the content of the dictionaries is strictly dependent of the application domain of the requirements document under evaluation.
- Multi-lingual: the modularity of the structure of the tool allows to analyze requirements documents written in different languages simply by changing the lower

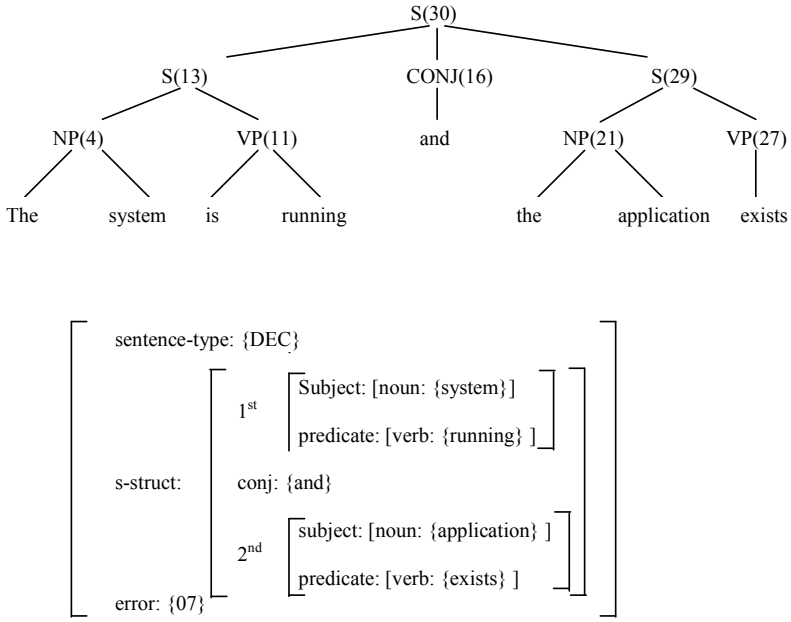

Figure 2. An example of derivation tree and the associated features structures

level components of the tool (lexical and syntactical analyzers and grammar methodologies) and by translating the dictionaries in the new target language. Table 3 provides some examples of requirements sentences containing defects detectable by using QuARS.

5. The QuARS tool in the requirements process: four case studies

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

- Input a requirements document in text format;
- Selection of an Indicator;
- Running QuARS;
- Using the output of QuARS (indication of those sentences in the requirements document containing potential defect according to the underlying quality model) in order to decide if modify the document.

In the following the above procedure is described more formally:
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.

INDICATOR	*NEGATIVE EXAMPLE*
Vagueness	• the C code shall be clearly commented
Subjectivity	• in the largest extent as possible, the system shall be constituted by commercially available software products
Optionality	• the system shall be such that the mission can be pursued, possibly without performance degradation
Implicity	• the above requirements shall be verified by test
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 minutes
Underreference	• the software shall be designed according to the rules of the Object Oriented Design
Unexplanation	• the handling of any received valid TC packet shall be started in less than 1 CUT

Table 3. Examples of requirement sentences containing defects

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

We have performed, by using QuARS, an analysis of real requirement documents taken from industrial software projects, in order to test the tool and understand if it may provide a real support to the improvement of the quality of NL requirements in an industrial environment.
The requirement documents that we have analyzed come 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: Requirements Specification of a project aiming for a new generation STM switches;
- $\mathcal{D}4$. Security Application: Functional Security Requirements for an Application Level Firewall Protection Profile.

We have calculated for each i_k of I the corresponding $\mathcal{RPD}^{i_k}_Q$. The outcomes of this kind of validation have been quite interesting. They are presented in the Table 4. that shows, for each evaluated document, the number of indicators' occurrences.
The number of elements of each $\mathcal{RPD}^{i_k}_Q$ may be then used to estimate the percentage of distribution of detected defect types. Figure 3. shows that some particular defects are more frequently detected than others by QuARS. In particular, multiplicity vagueness and under-specification seems to be the types of defects affecting a large part of the requirement sentences of a document.
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), are significantly high (around 50% of the total number of requirements sentences).

$\mathcal{D}1$	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. Numbers of warning messages occurrences for each Indicator

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 as actually defective. In other words, QuARS doesn't force the requirement engineers to follow a particular standard or style in writing but it should be considered as a support to point out potential weaknesses in the document which need careful management. An underlying syndrome affect our approach, it is the risk to point out too much defects in the requirement document under evaluation, in other words the risk to detect false positive. In fact, how can we be sure that all the defects detected and are real? The answer to this question depends strongly to the composition of the Dictionaries. If they are to short the risk to point out too few defects exists and on the other hand if they are too large is there the risk to indicate as defective also sentences that are not.

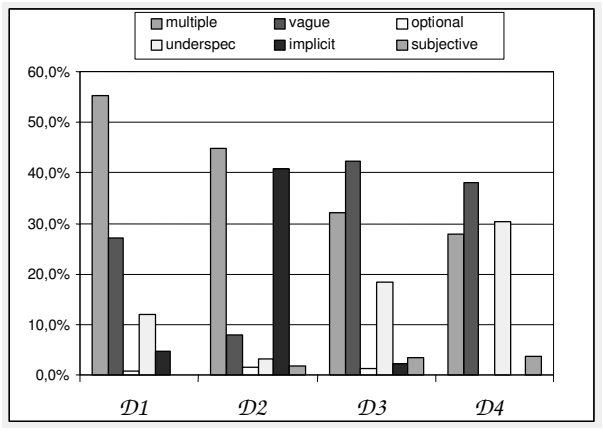

Figure 3.: Percentage distribution of warning messages types detected

A way to mitigate the effects of this syndrome is provided by the tool itself because it doesn't perform any automatic correction of the requirement sentences but it leaves the user free to accept the warnings or not on the basis of his knowledge of the application domain. Anyway, the mitigation of the false positive syndrome is up to the user too, because the more he is able to make the Dictionaries suitable to the specific application domain the more the results of QuARS will be precise.

6. Conclusions and Future Works

In this paper we presented a method and a tool 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, even though not exhaustive, 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 $\mathcal{RPD}_Q^K$ a significant set of actual defects occurring in the requirement specifications. The work carried out so far can be continued in several directions: to enlarge and refine the quality model in order to make the support provided more complete: 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. The proposed approach to the NL requirement quality can be applied also for the production of NL requirements by providing a guided editing of them.

Acknowledgements

This work has been partially supported by the project "CAFE, EUREKA 2023 Programme, ITEA project ip00004".

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

7. 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] N.E.Fuchs, R.Schwitter Specifying Logic Programs in Controlled Natural Language, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 3-5, 1995.

[5] 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.

[6] L.Goldin, D.M. Berry. 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.

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

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

[9] 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.

[10] G. Lami Towards an Automatic Quality Evaluation of Natural Language Software Specifications. Technical Report. B4-25-11-99. IEI-CNR, 1999.

[11] 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.

[12] 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.

[13] B.Meyer. On formalism in specifications. IEEE Software. January 1985, pages 6-26.

[14] 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.

[15] 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.
[16] N. Power Variety and Quality in Requirements Documentation Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, June 4-5 2001.
[17] 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.
[18] B.Welch Practical Programming in Tcl and Tk second edition Prentice Hall 1997.
[19] W.M.Wilson, L.H. Rosenberg, L.E. Hyatt. Automated quality analysis of Natural Language Requirement specifications. PNSQC Conference, October 1996.
[20] J.M. Wing, J. Woodcock, J. Davies (eds.) FM'99 – Formal Methods, vol. I and II LNCS 1708, 1709, Springer..