An Automatic Quality Evaluation for Natural Language Requirements

F. Fabbrini, M. Fusani, S. Gnesi, G. Lami
Istituto di Elaborazione dell'Informazione del C.N.R. – Pisa, Italy

Abstract

This paper presents a tool called QuARS (Quality Analyzer of Requirements Specification) for the analysis of natural language software requirements. The definition of QuARS has been based on a special Quality Model for software requirements. The Quality Model aims at providing a quantitative, corrective and repeatable evaluation of software requirement documents. To validate the Quality Model several real software requirements documents have been analyzed by our tool showing interesting results.

1. Introduction

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

- Natural language description of the functional and non-functional requirements of the software we are developing;
- Rigorous description of these requirements by means of formal/semi-formal methods [21], [17] (e.g. UML, SDL, etc.).

Our experience on the formal specification and verification of software requirements has shown that many problems occur in the passage from informal requirements, expressed in natural language, to semi-formal/formal models. These problems are mainly due to the inherent ambiguity originating from different interpretations of natural language descriptions. [8]
In order to reduce the risks associated to this passage, first it is necessary to have methods and tools to analyze natural language descriptions of the software/system under development. This analysis is aimed at detecting those problems that may affect the transformation of natural language requirements towards formal models.
Our purpose is to present an automatic tool for the analysis of natural language requirements documents to be used for detecting and removing defects that could cause errors in the transition to formal models.
The first step of our approach has been to define a Quality Model against which the requirements could be checked in order to remove ambiguities and incompletenesses. The Quality Model is composed of quality properties to be evaluated by means of quality Indicators.
Then a tool has been realized to implement the Quality Model and mechanize the analysis. The tool has been called QuARS (Quality Analyzer for Requirements Specifications).
We based our work on the study of natural language requirement documents taken from industrial projects. Moreover, we took into account several works in the domain of the quality analysis of natural language requirements [11], [13], [14], [15], [16]. In particular, Kamsties and Paech [10], starting from the consideration that ambiguity in requirements is not only a linguistic-specific problem, proposed a check list addressing not only linguistic ambiguity but also the ambiguity related

to the particular domain of the application under development. In their paper, Kamsties and Paech identified the principal deficiencies of the solutions to avoid ambiguity in natural language requirements:

1. Lack of acceptance by intended users and unfeasibility: extensive lists of rules are usually ignored;
2. Lack of specificity: current inspection techniques relying on checklist-based reading are unspecific. Usually checklists are ambiguous too.
3. Unidimentionality: they help to reveal linguistic ambiguities but not ambiguities respect other requirements, the application domain and the system domain.

The methodology we present in this paper, together with the associated tool, overcomes the deficiencies pointed out by Kansties and Paech since it is easy to use (1.), points out specific potential defects in the requirements document (2.) and detects not only linguistic ambiguities but also domain-dependent ambiguities (3.).

Also Wilson and others [20] face the quality evaluation of natural language software requirements. Their approach is similar to ours since they define a Quality Model composed of quality attributes and quality Indicators, and develop an automatic tool for performing the analysis against the Quality Model. Nevertheless, their work does not address some important issues as the adaptability of the tool to different domains and domains. Moreover, even if their Quality Model takes into account both syntactic and structural aspects of the requirements document, they don't consider the semantic aspects of the quality of requirements for their analysis.

Our tool includes and mechanizes all the aspects of the requirements quality considered in the two previous works, besides, it introduces the important issue of the adaptability/tailorability to different application domains of the quality analysis.

This paper is organized as follows: in Section 2. the Quality Model for natural language software requirements is described. In Section 3. the functionalities and the principal characteristics of the QuARS tool are described. In Section 4. we show how QuARS can be included into the requirement process. In Section 4. we describe the adaptability characteristic by means of an example of the adaptation to a particular application domain: the security domain. Finally, in section 5, conclusions are presented.

2. A Natural Language Software Requirements Specification (NLSRS) 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 adopted approach has been mainly top-down. In fact, starting from the identification of a set of representative and meaningful high level properties to be evaluated, we defined quality Indicators directly detectable and measurable on the requirement document.

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.
- *Completeness*: the capability of the requirements to refer precisely identified entities.

- *Understandability*: both 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.
- *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 but they are sufficiently specific for being applied (with the support of an automatic tool) to comparing and verifying the quality of requirement documents. Furthermore, this set of properties is sufficient to include a large part of the syntax-related issues of a requirement document along with a number of interesting semantics-related issues.

The above properties can be automatically evaluated by means of tangible Indicators. Indicators are syntactic or structural aspects of the requirement specification documents that provide information on a particular property of the requirements themselves. Several Indicators have been included in the Quality Model, each associated to a Property (see Table1.).

The Indicators can be counted/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.

For doing that both the single sentences of the requirement document and the whole requirement document structure are evaluated. 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). In other words, the quality Indicators of the Quality Model differ for their scope: some of them need to analyze single sentences for being calculated, others need to analyze the whole requirement document. Figure 1 shows that the object of the quality evaluation is the requirement document intended both as a whole and a set of sentences. The final quality evaluation is the union of the found defects regarding the requirement document structure and the single sentences.

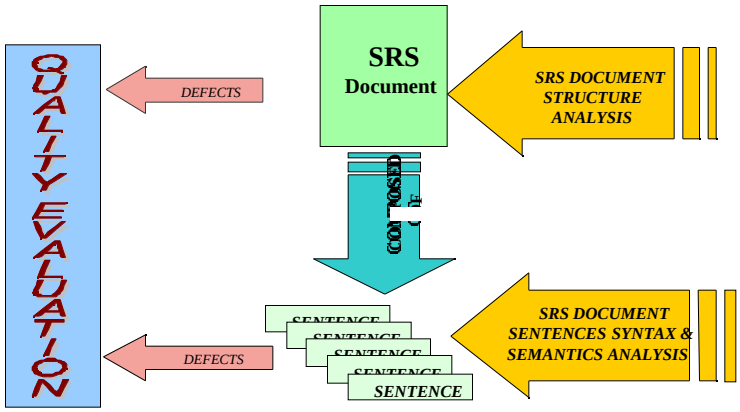

Figure 1. The quality evaluation of a requirement document

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 to be used for detecting potential defects in the NLSRS.

Table 2 provides some examples of requirements sentences containing defects detectable by using our Quality Model.

Table 3 shows the scope of each quality Indicator of the Quality Model.

The Indicator keywords used for detecting Indicators in the requirement document have been defined after the analysis of a number of requirement documents taken from industrial projects and on the basis of our experience in the Requirement Engineering [5], [6], [7], [12].

Property	Indicator	Description	Notes
	Optionality	An Optionality Indicator reveels 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, …*
	Subjectivity	A Subjectivity Indicator is pointed out if sentence refers to personal opinions or feeling	Subjectivitt-revealing wordings*: similar, better, similarly, worse, having in mind, take into account, take into consideration, as [adjective] as possible*
	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, efficient, useful, significant, adequate, fast, recent, far, close, in front, ...*
	Weakness	A Weakness Indicator is pointed out in a sentence when it contains a weak main verb	Weak verbs*: can, could, may.*
	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 modifier specifying an instance of this class	This indicator deals with the syntactic and semantics of the sentence under evaluation
	Under-reference	An Under-reference Indicator is pointed out in a NLSRS document when a sentence contains explicit references to: · not numbered sentences of the NLSRS document itself · documents not referenced into the NLSRS document itself · entities not defined nor described into the NLSRS document itself	-

Property	Indicator	Description	Notes
	Implicity	An Implicity Indicator is pointed out in a sentence when the subject is generic rather than specific.	Subject expressed by: Demonstrative adjective (*this, these, that, those*) or Pronouns (*it, they, ..*). Subject specified by: Adjective (*previous, next, following, last,..)* or Preposition (*above, below, ..*).
	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, …*
	Comment Frequency	It is the value of the CFI (Comment Frequency Index). [CFI= NC / NR where NC is the total number of requirements having one or more comments, NR is the number of requirements of the NLSRS document]	-
	Readability Index	It is the value of ARI (Automated Readability Index) [ARI=WS + 9*SW where WS is the average words per sentence, SW is the average letters per word]	-
	Directives Frequency	It is the rate between the number of NLSRS and the pointers to figures, tables, notes, ...	-
	Unexplana-tion	An Unexplaination Indicator is pointed out in a NLSRS document when a sentence contain acronyms not explicitly and completely explained within the NLSRS document itself	-

Table 1: Quality Properties and their Indicators

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
Under-specification	· 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.
Under-eference	· the software shall be designed according to the rules of the Object Oriented Design
Unexplaina-tion	· the handling of any received valid TC packet shall be started in less than 1 CUT

Table 2: Examples of requirement sentences containing defects

Quality Indicator	Requirement Document Sentences	Whole Requirement Document
Comment Frequency		•
Directives Frequency		•
Implicity	•	
Multiplicity	•	
Optionality	•	
Readability Index		•
Subjectivity	•	
Under- specification	•	
Under-reference		•
Unexplanation		•
Vagueness	•	
Weakness	•	

Table 3. Scope of the quality Indicators

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 is named QuARS and has been realized to parse requirement sentences written in Natural Language and to point out a number of potential sources of errors into the requirements themselves.
QuARS has been developed in the framework of a project[1] that aims at transferring technology to small and medium enterprises (SME) for improving their software development process.
In the tool we have developed the linguistic analysis engine defines a basic English grammar (about 40 production rules, currently) and a small dictionary.
The dictionary contains a set of grammatical words (such as determiners, particle, quantifier, auxiliary verbs, etc.) manually developed by a computational linguist and a set of "semantic" words (nouns, adjectives, adverbs, verbs) automatically generated by means of a the morphological analyser ENGLEX (http://www.sil.org) running on our test corpus. The dictionary contains also the list of words defined by the AECMA-Boeing Simplified English Project (http://www.aecma.org/Publications/SEnglish/senglish.htm).
For each word, part of speech and morphological information are stored; grammatical words can also have a grammatical functions defined.
The grammar covers basic structures of main sentence, direct question, infinitive clauses, noun phrases and verb groups.

Summarizing, the QuARS tool is composed of the following main logic modules:

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

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

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

- The files containing the NLSRS 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. It, 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 figures 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 special Dictionaries as input. These Dictionaries contain the words and the syntactical elements that allow the detection of inaccuracies in the NLSRS sentences (see for example the Notes column in the 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 those sentences of the NLSRS Document having potential defects.

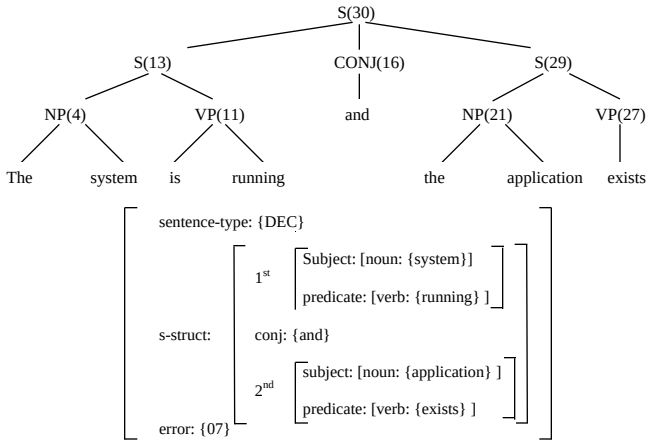

Figure 2: An example of derivation tree and the associated features structures

Figure 3. depicts the operation of the QuARS tool.

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.
- Generality: the tool shall 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

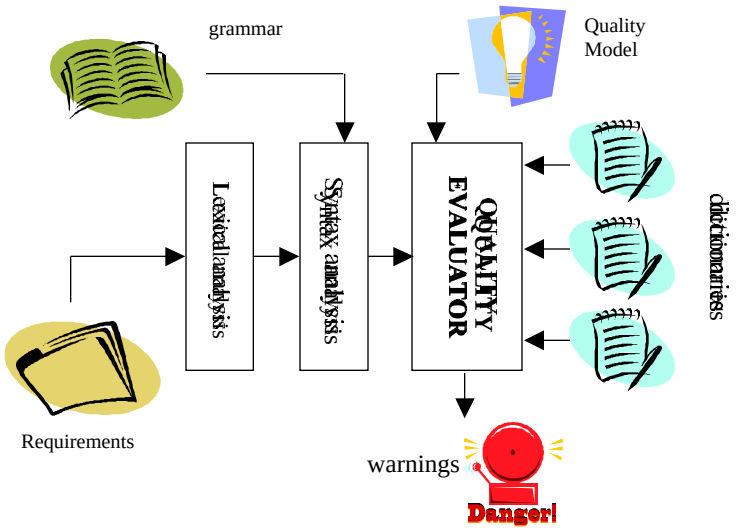

Figure 3 Scheme of the QuARS tool operation

In the following we will describe how the main target characteristics (ease to use, generality and flexibility) have been implemented.

Ease to use

Despite of QuARS has its own text interface, a TCL/TK [2] graphical interface has been released to simplify the use of the tool and to enhance his characteristics (see figure 4). The script language TCL/TK has been used because of his portability and the very fast way to develop, verify and debug the code: this fits good with the building a not integrated interface for a tool with his own independent behavior and environment.

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

· to do an easy and fast navigation in the file system for selecting the file to analyze,
· to edit and save the specification file to analyze,
· to remember the latest working directory,
· to edit and save all the QuARS dictionaries using the simple but complete built-in editor,
· to easy choose the work session to perform,
· to save into file the results clearly visualized in the wide central window for a future use,
· to see the statistics about the current session work,
· to illustrate it, an on line help, the analysis purpose and the functionality of the graphical interface itself.

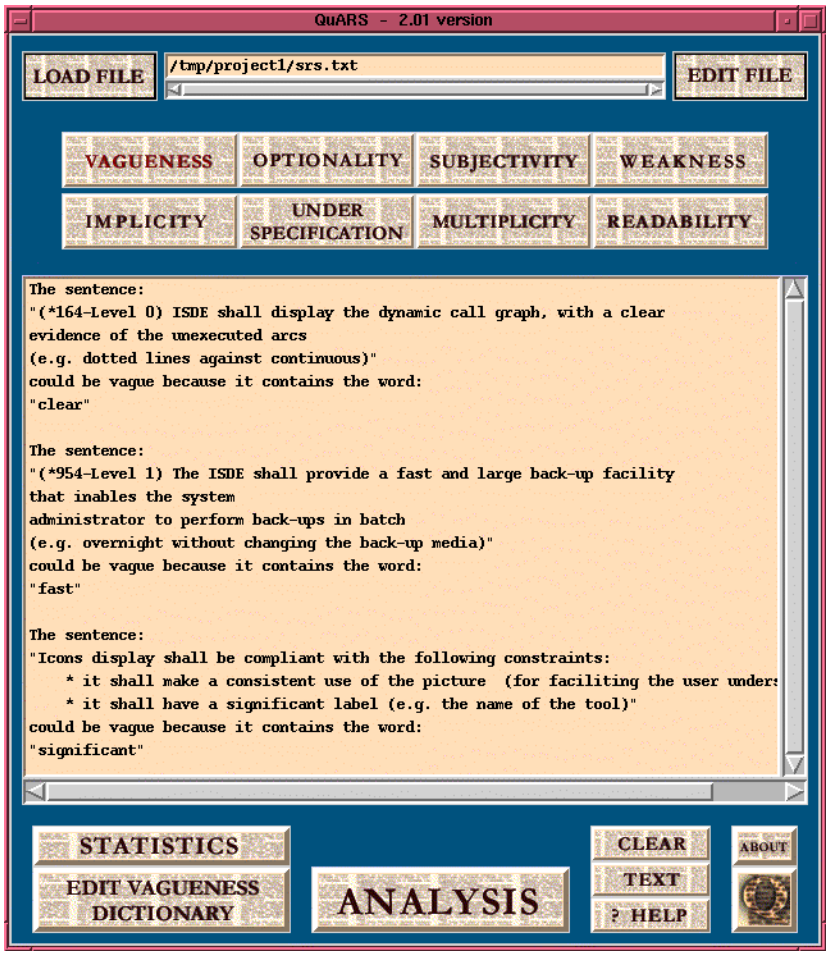

Figure 4: QuARS: the user interface

Generality

The expected format of the requirements sentences to be evaluated is the text format.

This format allows to achieve an high generality because it is always possible to save every format as text format.

The risk associated to this strategy is to left 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 every requirement documents the hierarchy of the requirements isn't be established by means of the formatting of the text, but it has to be defined by means of appropriate numeration of the sentences. Then the possible lack of information doesn't compromise the validity of the textual requirement document.

Flexibility/Tailorability
The flexibility/tailorability target characteristic is very important for the use of QuARS in industrial domain. In fact, it is important 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 several Indicators, and in some cases the content of the dictionaries is strictly dependent of the application domain of the requirements document under evaluation. Some Indicators are most dictionary-sensitive than others (e.g. Vagueness and Under-specification)

For these Indicators, the possibility to adapt the related dictionary is necessary to assure a meaningful evaluation.

4. The QuARS tool in the Requirements Process

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.
A typical evaluation of a requirement document is performed by a team of expert reviewers who, by analyzing the NLSRS documents, could detect ambiguities, inconsistencies or, in general, inaccuracies. Other actors may be involved in setting up the requirement document, but at evaluation (review) time the reviewer is the key actor.
QuARS can be helpfully included in the software requirements production and quality evaluation process (see figure 5).

The QuARS tool, for its nature, can be tuned up in order to make it more adequate to particular application domains or types of software requirements. Then the inclusion of the QuARS tool in the software requirements quality evaluation process allows the reviewer to change the Quality Model if necessary in order to make it more respondent to the particular NLSRS under evaluation. The conceptual changes on the Quality Model can be easily transferred on QuARS for its flexibility. Finally, the corrective actions are made on the NLSRS document and the process can be re-started on the modified document. When the requirement document passes the judgment of the Requirements Quality Evaluation Team it will be approved and injected in the development process.
We run QuARS on real requirement documents taken from industry software projects.
The requirements documents run with the QuARS tools are given from different application domains:

S1. Business Application: Functional Requirements of a Transaction and Customer Service (TACS) Check Cashing module;
S2. Space Software Application: Functional Requirements of a sub-system of a space vehicle;
S3. Telecommunication Application: Requirements Specification of a project aiming for a new generation STM switches;
S4. Security Application: Functional Security Requirements for an Application Level Firewall Protection Profile

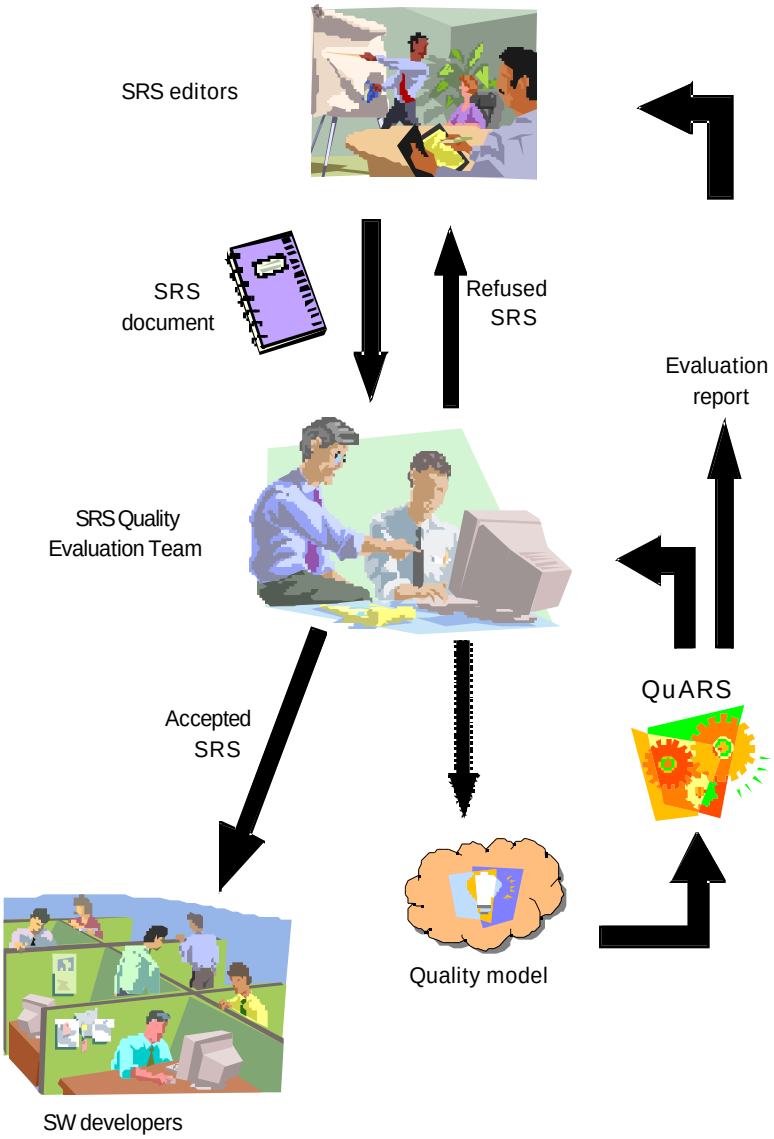

Figure 5: The use of QuARS into the NLSRS process

The outcomes of this kind of validation have been interesting. They are shown in the Figure 6 and Figure 7. Figure 6 shows the rate of defects occurrences on the total requirements for each evaluated documents. Figure 7 shows the percentage distribution of defects types detected for each of the four requirements evaluated.

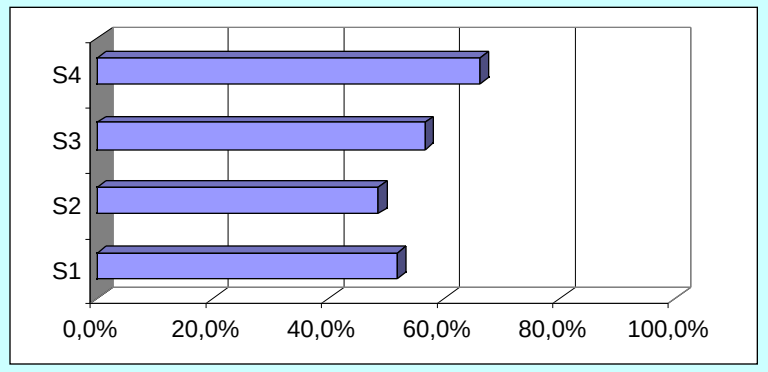

Figure 6. Rate of defects occurrences on the total requirements

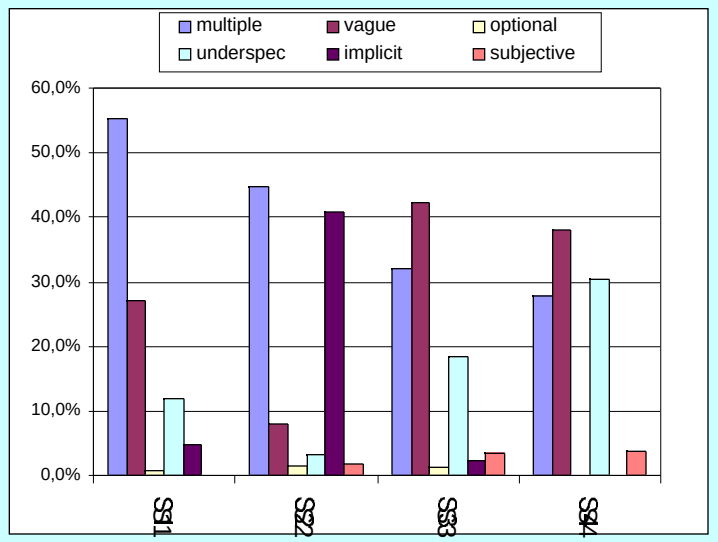

Figure 7: Percentage distribution of defects types detected

What we can note is, first, that the rate of defects detected is at least the 50% for each requirement document evaluated. It means that the Quality Model (and the QuARS tool that implements it) can be considered effective in defect detection. It is also possible to observe that the multiplicity and vagueness and under-specification Indicators are most frequently pointed out.
Requirements engineers from the involved industries found these results useful and interesting for improving their requirement documents

5. An Example of Tailoring of the QuARS tool: Analysis of Software Security Requirements

Before to perform an evaluation of a new requirement document it is necessary to adjust the QuARS tool in order to make it more suitable for evaluating the new requirements.
In order to verify if the expected flexibility of this tool is really applicable to requirements dealing with a particular application domain, QuARS has been used for evaluating requirements dealing with a particular application domain: security applications.
With this aim some Special Dictionaries of QuARS have been reviewed and modified in order to make them more suitable to an environment where IT Security requirements are essential and then to make QuARS more effective for this environment.
The changes made for tailoring the QuARS tools regarded in particular the dictionary associated to the Under-specification Indicator. Some special words, indicating classes of objects that in a security environment needed a modifier for being uniquely specified, have been added.
The security-tailored QuARS tool has been then used for analyzing the S1. Requirements Document "U.S. Government Application-Level Firewall Protection Profile for Low-Risk Environments" [19] document, that defines the basic security requirements of U.S. Government organizations handling unclassified information in low-risk environment. This analysis aims to check if this document, that is conform to Common Criteria Part 2 and Part 3 [3], contains lacks of quality according to our Quality Model that can cause errors in the subsequent phases of the software development.
The changes made for adapting QuARS to a security domain consist in the addition of special words in the Under-specification-related dictionary.
In the following a sample of the security-related words added to the Under-specification dictionary is provided:

Key
This word in a security environment means: "a long string of seemingly random bits used with cryptographic algorithms to create or verify digital signatures and encrypt or decrypt messages and conversations. [1]
Nevertheless, in this environment it is necessary to specify which type of key we are considering. In fact, different types of key exist: private key, public key, secret key, etc.

Access
This word in a security environment means: "A specific type of interaction between a subject and an object that results in the flow of information from one to the other. [4]
Nevertheless, in this environment it is necessary to specify which type of access we are considering. In fact, different types of access exist: write access, remote access, authorized access, unauthorized access, etc.

Security policy
It means: "A set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information". [18]
Nevertheless, in this environment it is necessary to specify the domain a security policy is referred to. In fact, different security policy domains may exist: Corporate security policy, System security policy, Technical security policy.

Attack
This word in a security environment means: “The act of trying to bypass security controls on a system. An attack may be active, resulting in the alteration of data; or passive, resulting in the release of data.” [18].
Nevertheless, in this environment it is necessary to specify which type of attack we are considering. In fact, different types of attack exist: physical attack, simple attack, complex attack, etc.

It is now possible to understand why the under-specification of these words can cause bad interpretation of the requirements and then introduce risks in the project. This is the reason why the QuARS tool points out the occurrences of these words when they have a loss of specification.

5. Conclusions

In this paper we have present a method for evaluating natural language software requirements according to a previously defined Quality Model.
The evaluation of requirements documents made 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 has been defined with the purpose to detect and point out potential syntactic and semantic deficiencies that can cause problems when a natural language requirement document is transformed in a more formal document. The definition of the criteria used in the Quality Model has been driven by some results in the natural language understanding discipline (in order to detect the syntactical components introducing for example ambiguity), by our experience in formalization of software requirements and also by a depth study of real requirements documents provided us from industrial partners. Indeed they were involved during the definition of the Quality Model to give us their impression about our choices in order to better understand the validity of the proposed model.
After the definition of the Quality Model against which to evaluate natural language requirement documents, a tool, named QuARS (Quality Analyser for Requirements Specifications), based on the developed model has been developed.
In order to achieve confidence about the effectiveness of our method/tool, four industrial requirement documents written in natural language has been evaluated by QuARS. The outcomes has been encouraging because the tool found a large number of potential defect and the requirements engineers of the involved industries have judged useful the results obtained to improve their documents.

6. References

[1] M. Abrams, S. Jajodia, H. Podell, eds, Information Security – An integrated Collection of Essays, IEEE Computer Society Press, January 1995.
[2] Practical Programming in Tcl and Tk” second edition Prentice Hall 1997.
[3] Common Criteria for Information Technology Security Evaluation, CCIB-98-026 Version 2, May 1998.
[4] Dept. Of Defense Standard, Department of Defence Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, GPO 1986-623-963, Dec. 26, 1985.

[5] 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.

[6] 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.

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

[8] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P.Moreschini, "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.

[9] S.Gnesi, D.Latella, G.Lenzini, A "Brutus" Model Checking of Spi-Calculus Dialect, Proc.of the Workshop on Formal Methods and Computer Security, FMCS 2000, 2000.

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

[11] 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.

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

[13] F.Lehner. Quality control in software documentation based on measurement of textcomprehension and text comprehensibility. Information Processing & Management, vol; 29, No. 5, pp 551-568, 1993.

[14] B.Macias, S.G. Pulman. Natural Language processing for requirements specifications. In Redmill and Anderson, *Safety Critical Systems*, pages 57-89. Chapman and Hall, 1993.

[15] M.Mannion, B.Keepence, D.Harper. Using viewpoints to define domain requirements. IEEE Software, January-February 1998, pages 95-102.

[16] K.Matsumura, H.Mizutani, M.Arai. An application of structural modelling to software requirements analysis and design. IEEE Transactions on Software Engineering, vol. SE-13, No.4, April 1987.

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

[18] Nat'l Computer Security Center, Trusted Network, Glossary of Computer Security Terms, NCSC-TG-004, Oct. 1988.

[19] U.S. Government Application-Level Firewall Protection Profile for low-Risk Environment, Version 1.b, September 1998..

[20] W.M.Wilson, L.H. Rosenberg, L.E. Hyatt. Automated quality analysis of Natural Language requirement specifications. PNSQC Conference, October 1996.

[21] J.M. Wing, J. Woodcock, J. Davies (eds.) FM'99 – Formal Methods, vol. I and II LNCS 1708, 1709, Springer.