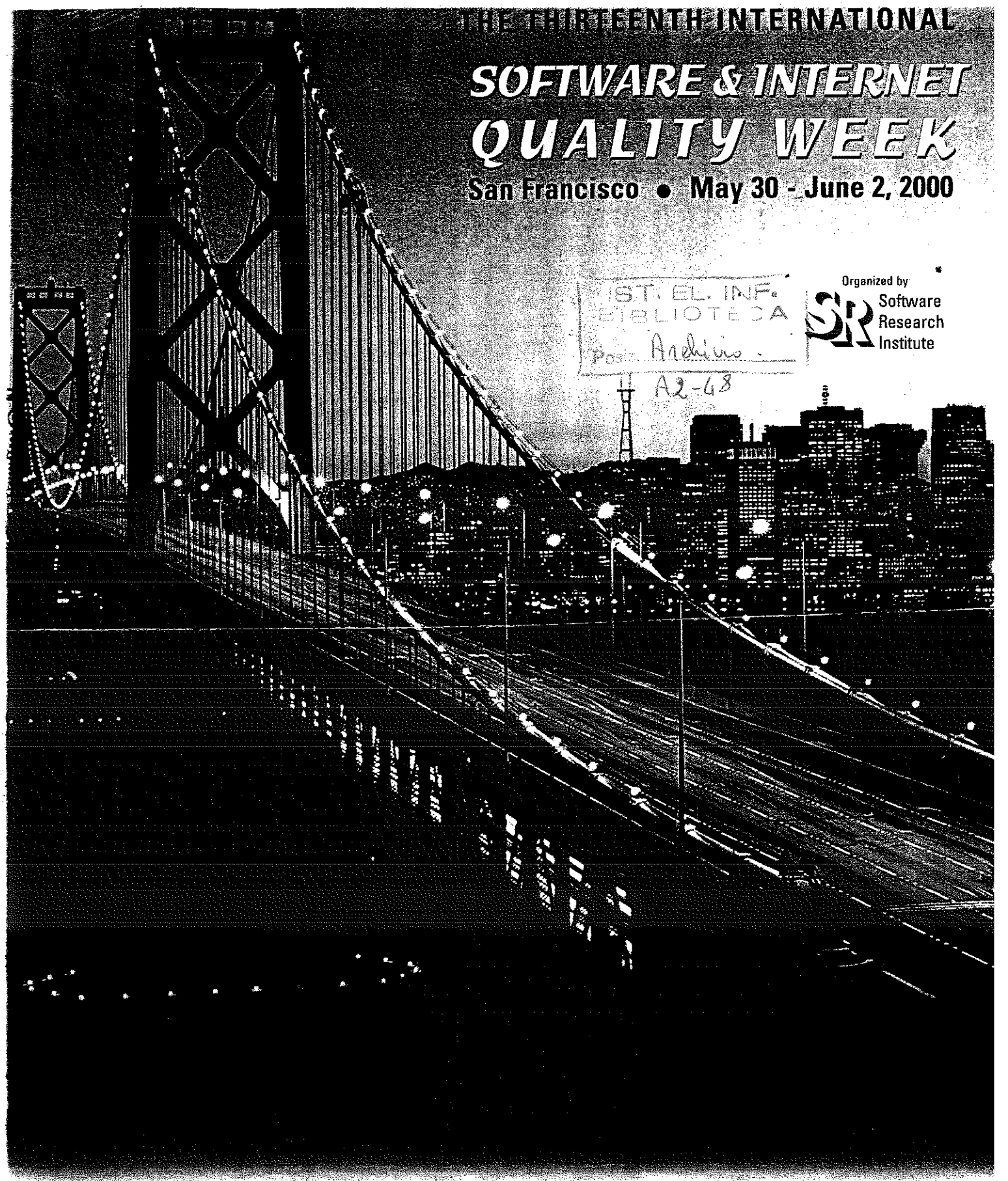


THE THIRTEENTH INTERNATIONAL

SOFTWARE & INTERNET QUALITY WEEK

San Francisco • May 30 - June 2, 2000

Organized by
Software Research Institute

In cooperation with

acm

American Society for Quality
ASQ
Software Division

Industry Sponsors

COMPUWARE.

Rational
the e-development company

Valid
Your e-Business Partner

SOFTWARE RESEARCH

Microsoft
Where do you want to go today?

VANTEON

NASXXCAL
TESTING LABORATORIES

ZD LABS

Media Sponsors

SOFTWARE Development

CUTTER CONSORTIUM

Software BUSINESS

Quality Evaluation of Software Requirements Specifications

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

I.E.I. - C.N.R. Pisa, Italy

The criticality of the Software Requirements Specifications (SRS) phase of the software life cycle for the success of the whole software project is widely recognized and the attention played on it by software developers is more and more significant. The software science developed in the past methods and techniques for producing quality SRS: structured languages, controlled languages, formal languages.
Nevertheless the SRS phase is, especially in the Small and Medium Enterprises (SME) community, still one of the weakest steps of the whole software process. One of the principal reasons of this situation is that SMEs, stressed by the time-to-market, do not have enough resources for establishing rigorous methods for the SRS. Furthermore such rigorous methods should be shared among all the parties involved in a software project, customer included, asking for a further effort in terms of resources to be expended.
It is not surprising then that Natural Language (NL) SRS are, in spite of their inherent inaccuracies, still the most used technique for SRS.
In this paper we describe the effort made at I.E.I - C.N.R. for realizing an automatic tool (called QuARS - Quality Analyzer of Requirements Specifications) supporting the analysis and quality evaluation of NL SRS. The adopted approach has been first to define a Quality Model for SRS, then to verify this Model on real cases of SRS in order to be sure that the Quality Model provides a real contribution for solving some NLSRS related problems for SMEs. Finally we have developed an automatic tool for the verification of the requirements to guarantee their conformance to the Quality Model itself.
This paper is organized as follows: in Section 1 we provide general considerations on Software Requirements Specification by considering NL techniques and other more formal techniques. In Section 2 we describe the proposed Quality Model for NL SRS. In Section 3 we describe the outcomes of the experimentation of the proposed Quality Model on real industrial cases. In Section 4 we describe the functionalities of the QuARS tool. In Section 5 we focus on the Quality Evaluation Process of NL SRS, and, finally, in Section 6 we discuss conclusions and future works.

* This work was partially supported by SSSUP S.Anna in the framework of the LINK project, MURST L. 488/92.

1. Techniques for producing Software Requirements Specifications

The Requirements Engineering is the process of establishing the services the system should provide and the constraints under which it must operate [13].

Ideally the Requirements Engineering process should be separated into three sub-processes, each needing different styles, levels of detail and people to be involved. The separation of these processes should guarantee the production of a suitable final output toward the software designer. The next table summarizes the principal aspects of these processes [13]:

Requirements Engineering phase	Level of Detail	People involved
Requirements Definition: statement, in a natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate. It is generate using customer-supplied information.	General	Managerial Level End-users System architects Client engineers
Requirements Specification: a structured document which sets out the system services in detail. This document should be precise. It may serve as a contract between the system buyer and software developer	Intermediate	End-users System architects Client engineers SW developers
Software Specification: abstract description of the software which is a basis for design and implementation.	Highly detailed	System architects SW developers

The expected outputs of the previous phases are documents that gradually fill the gap between the conceptual understanding of the system we want to build and the detailed description of its functionalities that have to be fed to the system architects.
The phase of the Requirements Engineering Process addressed in this paper is the Requirements Specification.

The scheme presented before is an ideal scheme and often it doesn't correspond to the practices of software firms. In fact, the real software process of many firms considers the Requirements Engineering process as composed of an unique phase aimed to provide a document describing, in some way, the Requirements of the system to be developed.
There are several risks associated with this practice, as, for example, those listed below:

- The requirements related to different functionalities of the system may be not expressed with the same level of detail and precision;
- The requirements may be too abstract and then not easily usable by the software designers.
- The involvement of some people in the Requirements Engineering process may be not fully adequate (e.g. software developers, end-users, ..).

We will focus our attention on a product that we can assume will be anyway provided as an outcome of the Requirements Specifications phase: a document describing somehow and with a certain level of precision the functionalities and the constraints of the system under construction.

Several approaches to producing Requirements Specification exist and their precision and cost may vary considerably. We will concentrate our attention on one of these approaches: the use of Natural Language for writing Requirements Specification. Even though it is out of the scope of this paper to compare weaknesses and strengths of different Requirements Specifications approaches and to establish if one of them is better than another, in the next scheme we provide a brief description of the most used approaches.

Approach to Requirements Specifications	*Description*
Natural Language	-
Structured Natural Language	restricted natural language where the terminology is limited and templates can be used. Control constructs derived from programming languages can be included.
Semi-formal Languages	they are usually special-purpose graphical notations with a precise syntax and a non-rigorous semantic.
Formal Languages	mathematics based languages with vocabulary, syntax and semantics formally defined.

It is clear that the effects of the use of each of them on the quality of the Requirements Specifications are different and it is also quite clear that the order in which they are listed into the scheme correspond to an increasing "quality" of the Requirements Specification they produce. In other words, it is clear that Requirements Specifications written by using NL may contain ambiguities, while SRS written by using Formal Methods are more precisely and rigorously understandable.
However, the establishment of a formal method for the SRS needs some initial additional costs. These costs are due to the training of people in order to make them able to use the formal techniques. In other words it is necessary to make all the people involved in the SRS phase, often coming from different social, business and technical contexts, able to share these formalisms.
In spite of these considerations, many SMEs, even though they recognize both the importance and the benefits of the use of formal methods, and the risks due to the use of NL in writing SRS, continue to write SRS using natural languages unless they are not obliged to do otherwise by precise contractual constraints.

Then the fact that writing SRS in NL is not a good practice because they are prone to ambiguities and that they are neither a good input for the design phase nor a good basis for the contractual issue between the customer and the supplier doesn't mean that we have to renounce achieving an acceptable level of quality even in NL SRS. On the contrary we are persuaded that it is useful to have a pragmatic approach to the question and then it is a worth-while effort to provide a contribution in the quality improvement of NL SRS. This approach has been adopted in the definition of the Quality Model and in the consequent realization of the QuARS prototype.

2. A NLSRS Quality Model

The Quality Model defined in this paper aims to provide the means for evaluating quantitatively two aspects of the SRS quality, called Goal Properties:

- Requirements Sentences Quality (RSQ): the syntactical quality of single sentences considered separately; [4]
- Requirements Document Quality (RDQ): the quality of the sentences considered in the context of the whole requirements documents.

These two Goal Properties are obviously too abstract for being directly evaluated. For this reason some Properties have been associated to each Goal Property in order to provide quality characteristics less abstract and then easier to evaluate. Table 1. shows the Quality Model Goal Properties and the related Properties:

Goal Properties	*Properties*
Requirement Sentences Quality (RSQ)	Non-Ambiguity
	Completeness
	Understandability
Requirement Document Quality (RDQ)	Completeness
	Understandability

Table 1.

The Properties related to the RSQ Goal Property included within our Quality Model are:

- Non-Ambiguity: the capability of a Requirement to have a unique interpretation.
- Completeness: the capability of each Requirement to make reference to precisely identified entities.
- Understandability: the capability of each Requirement to be fully understood when used for developing software.

The Properties related to the RDQ Goal Property included within Quality Model are:

- Completeness: the capability of the Requirements Specification Document to avoid potential or actual discrepancies.
- Understandability: the capability of the Requirements Specification Document to be fully understood when read by the user.

The above Goal Properties and Properties have been included into the Quality Model because they can be automatically evaluated by means of tangible Indicators that can be counted/detected during the parsing of the requirements document, and because possible problems related to these Properties can be pointed out precisely so that corrective actions may be easily done.

This Quality Model is focused on linguistic properties of SRS and cannot be considered complete because other aspects of the SRS quality, as for example semantic consistency aspects, exist and they are not taken into account.

2.1 Quality Indicators

Quality Indicators are syntactic aspects of the requirements specifications [14] that can be automatically calculated and that provide information on a particular quality Property of the requirements specifications themselves. Several Indicators have been included in the Quality Model, each associated to a Property.
In the following, the Indicators are listed and described, then the association between them and the Properties of the Quality Model is shown.

The Quality Indicators included in the Quality model are divided into Indicators related to Requirement Sentences Quality (RSQ) and Requirement Document Quality (RDQ):

Requirement Sentences Quality (RSQ) related Indicators:

- Implicit Subjects Sentences
- Multiple Sentences
- Optional Sentences
- Subjective Sentences
- Underspecified Sentences
- Vague Sentences
- Weak Sentences

Requirements Document Quality (RDQ) related Indicators:

- Comment frequency,
- Readability Index
- Underreferenced Sentences
- Unexplained Sentences

In the following the definition of each Quality Indicator is provided along with some examples.

Implicit Subject Sentences:

A sentence is an implicit subject sentence if its subject:

- contains a demonstrative adjective: this, these, that, those
- is expressed by means of pronouns: it, they.
- is specified by a preposition as such: above, below, ...
- is specified by an adjective as such: previous, next, following, last, first, ...

Examples:

Implicit Subject Sentences:

- This counter shall be incremented by 1 each time a RTI occurs.
- It shall be stored in non-volatile memory.
- The above requirements shall be verified by test.

Non-implicit Subject Sentences:

- The MC counter shall be incremented by 1 each time a RTI occurs.
- The detected erroneous data shall be stored in non-volatile memory.
- The requirements number 1 to 56 shall be verified by test.

Multiple Sentences:

A Sentence is multiple if:

- it has more than one subject or more than one main verb
- it has more than one direct or indirect complement that specifies its subject

Examples:

Multiple sentence:

- The System shall provide for the generation of stub/driver skeleton and for the simulation of module interface.
- The customer perception of System Down Time and Partial System Down shall be zero.

Non-multiple sentences:

- The System shall provide for the generation of stub/driver skeleton.
- The System shall provide for the simulation of module interface.

- The customer perception of System Down Time shall be zero.

- The customer perception of Partial System Down shall be zero.

Optional Sentences:

A sentence is optional if it contains an option .

Note: A sentence is to be considered as optional if it contains words as such: possibly, eventually, if appropriate, if needed .....

Examples:

Optional sentence:

- The system shall be such that the mission can be pursued, possibly without performance degradation.

Non-optional sentence:

- The system shall be such that the mission can be pursued, with performance degradation not greater than 10%.

Subjective Sentences:

A sentence is subjective if it refers to personal opinion or feeling.

Note: sentences containing the following kinds of wording are to be pointed out as potentially subjective:

- having in mind, take (into) account, take into consideration, ...
- similar, better, similarly, worse, ...
- as [adjective] as possible.

Examples:

Subjective Sentences:

- The System should allow computation analysis results to be presented on-line during the coding phase (having in mind the basic set of metrics).
- Software tasks shall be as much as possible synchronous.

Non-subjective Sentences:

- The System should allow computation analysis results to be presented on-line during the coding phase by showing values of the basic set of metrics.
- Software tasks shall be synchronous.

Underspecified Sentences:

A sentence is underspecified if its subject contains a word identifying a class of objects without a modifier specifying an instance of such class.

Examples:

Underspecified Sentence:

- The testers shall be independent of the software developers.

Non-underspecified Sentence:

- The module testers shall be independent of the software developers.

Vague Sentences:

A sentence is vague if it includes words holding inherent vagueness.

Note: Vague adjectives are:

- adjectives related to intrinsic characteristics as such: clear, well, easy, strong, weak, good, bad, efficient, low .....
- adjectives related to environmental characteristics as such: useful, significant, adequate, fast, slow....
- adjectives related to time characteristics as such: old, new, future, recent, past, today's, ...
- adjectives related to location characteristics as such: near, far, close, back, in front, ...

Examples:

Vague Sentences:

- The C code shall be clearly commented.
- The User Manual shall provide adequate information about the installation procedure.
- The Reports shall be produced according to the today's trend.

Non-vague Sentences:

- The sentences of the comments into the code shall be no longer than 20 words.
- The User Manual shall provide step by step procedure for the installation.
- The Reports shall be produced according to the template provided in figure 1.2.

Weak Sentences:

A sentence is weak if it contains a weak verb.

Note: Weak verbs are:

- can / could
- may

Examples:

Weak Sentence:

- The results of the Initialization checks may be reported in a special file.

Non-weak Sentence:

- The results of the Initialization checks shall be reported in a special file.

Comment Frequency:

The Comment Frequency of a Requirements Document 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 total number of Requirements of the document)

Underreferenced Sentences:

A sentence of a Requirements Document (RD) is underreferenced if it contains explicit references to:

- not-numbered sentences of the RD itself,
- documents not-referenced into the RD itself,
- entities not defined nor described within the RD itself.

Note: The following kinds of wording are to be considered as explicit references:

- *according to*
- *on the basis of*
- *relatively to*
- *compliant with*
- *conformant to*

Example:

Underreferenced Sentence:

- The Software shall be designed according to the rules of the Object Oriented Design

Non-underreferenced Sentence:

- The Software shall be designed according to the rules contained in [TD.4]. (where [TD4] is included into the list of references of the same Requirements Document).

Unexplained Sentences:

A sentence of a Requirements Specification Document (RSD) is unexplained if it contains acronyms not explicitly and completely explained within the RSD itself.

Examples:

Unexplained Sentences:

- The handling of any received valid TC packet shall be started in less than 1 CUT.
- The results of the Initialization checks shall be reported in the HK TM.

Readability Index:

The unreadability index is the value of ARI (Automated Readability Index) [6] (ARI=WS + 9SW where WS is the average words per sentence, SW is the average letters per word)

Each of the above Quality Indicators has been associated with a Property and the Quality Model has been then completed. The table 2 shows the Quality Model obtained by including the previously defined Quality Indicators.

Goal Properties	Properties	Quality Indicator
Requirement Sentences Quality	Non-Ambiguity	Implicit Subjects Sentences, Optional Phrases, Subjective Sentences, Vague Sentences, Weak Sentences
	Completeness	Underspecified Sentences
	Understandability	Multiple Sentences
Requirement Document Quality	Consistency	Underreferenced Sentences
	Understandability	Comment Frequency, Readability Index, Unexplained Sentences

Table 2.

2.2 Quality Requirements for Software Requirements Specifications

The Quality Indicators can be automatically detected and counted during the analysis of the requirements specification document. The results of this analysis have to be provided in order to make the user able to point out the sentences that are potentially incorrect. Then the user, on the basis of his experience and skill has to judge if the pointed out sentences are really incorrect or if they can be accepted by taking into account the following quality requirements.

The quality requirements for each Indicator are listed below.

A Requirements Document shall not contain:

- Implicit Subjects Sentences ;
- Multiple Sentences;
- Optional Sentences;
- Subjective Sentences;
- Underspecified Sentences;
- Vague Sentences;
- Weak Sentences;
- Underreferenced Sentences;
- Unexplained Sentences;

A Requirements Document shall:

- have the Comments frequency value greater than an established target value X*;
- have the Readability Index value greater than an established target value Y*.

*: the target values determination is project dependent.

3. Validation of the Quality Model

In order to verify if the defined Quality Model provides significant and useful results on real cases, it has been applied to two different sets of documents of real industrial requirements specifications.
The first is a set of Telecommunication Software Requirements Specification documents, the second is a case of Safety Critical Space Software Requirements Specification documents. In both cases the organizations that produced these requirements specifications use standards for writing them, perform review sessions and produce the related reports.
The results achieved are shown below:

Telecommunication Software Requirements Specification documents analysis and evaluation results

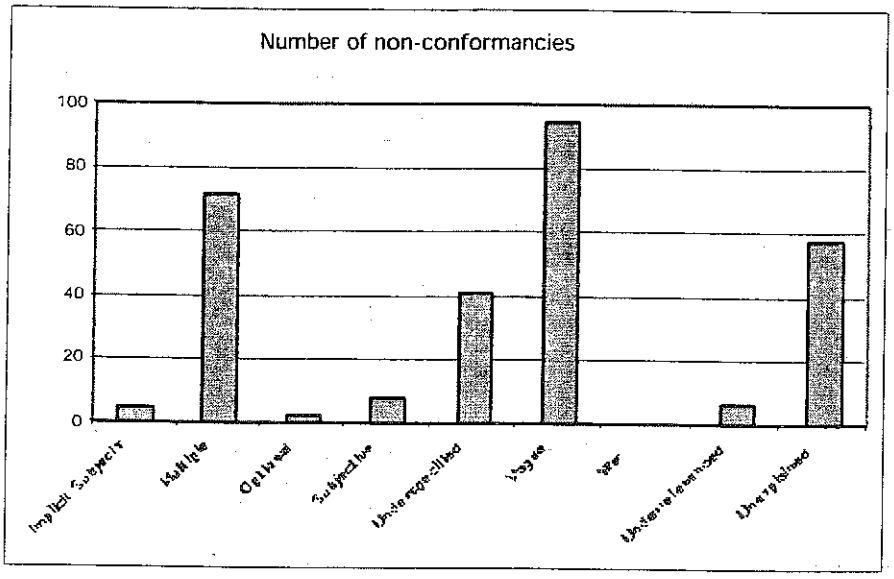
Indicators taken into account:

a) Implicit Subjects Sentences
b) Multiple Sentences
c) Optional Sentences
d) Subjective Sentences
e) Underspecified Sentences
f) Vague Sentences
g) Weak Sentences
h) Underreferenced Sentences
i) Unexplained Sentences

Total number of evaluated requirements: **395**

The following table shows the number of requirements that don't satisfy the quality requirements:

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	5	1,3%
Multiple Sentences	72	18,2%
Optional Sentences	3	0,8%
Subjective Sentences	8	2%
Underspecified Sentences	41	10,4%
Vague Sentences	95	24,1%
Weak Sentences	0	0%
Underreferenced Sentences	7	1,8%
Unexplained Sentences	58	14,7%



Faults rate = 73%
Note: Faults rate = (number of non-conformances) / (number of requirements)

Safety Critical Space Software Requirements Specification documents analysis and evaluation results:

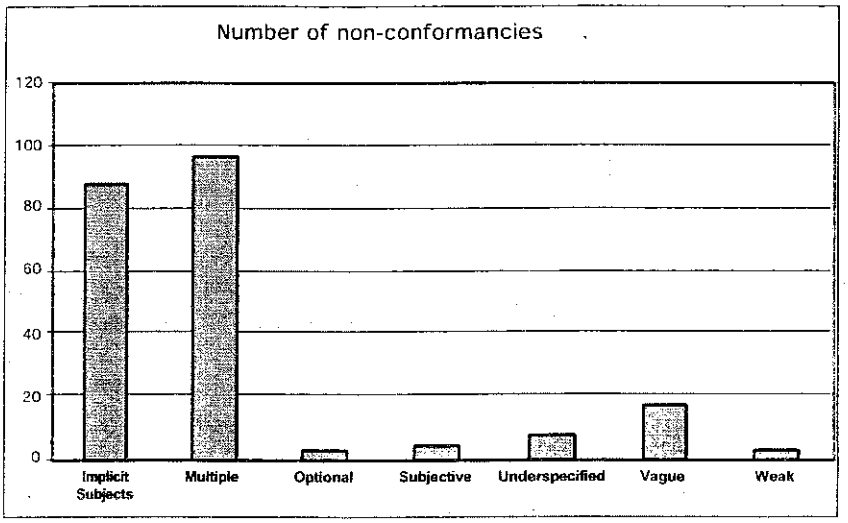
Indicators taken into account:

a) Implicit Subjects Sentences
b) Multiple Sentences
c) Optional Sentences
d) Subjective Sentences
e) Underspecified Sentences
f) Vague Sentences
g) Weak Sentences

Total number of evaluated requirements: **444**

The following table shows the number of requirements don't satisfy the quality requirements:

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	88	19.8%
Multiple Sentences	97	21,8%
Optional Sentences	3	0,7%
Subjective Sentences	4	0,9%
Underspecified Sentences	7	1,6%
Vague Sentences	17	3,8%
Weak Sentences	3	0,7%



Faults rate = 49%

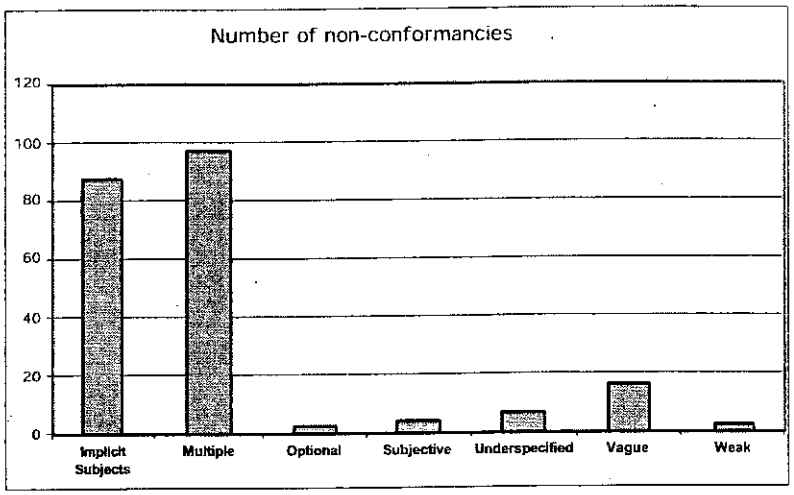
note: Faults rate = (number of non-conformances) / (number of requirements)

Joint results:

Total number of evaluated requirements: **839**

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	93	11,1%
Multiple Sentences	169	20,1%
Optional Sentences	6	0,7%
Subjective Sentences	12	1,4%
Underspecified Sentences	48	5,7%
Vague Sentences	112	13,3%
Weak Sentences	3	0,4%
Underreferenced Sentences	7**	
Unexplained Sentences	58**	

**: *calculated only for the Telecommunication Software Requirements Specification documents.*



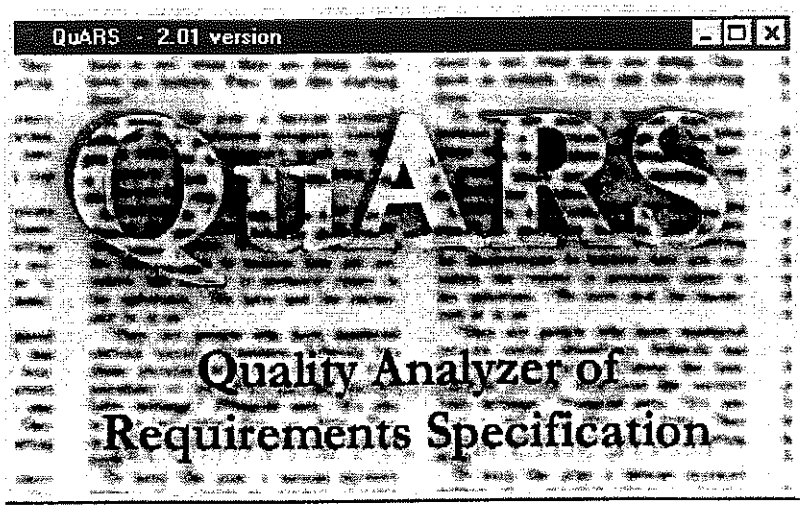
Faults rate = 53%

note: Faults rate = (number of non-conformances) / (number of requirements)

The firms that provided the analyser with the requirements were presented these results in order to achieve their comments and, in particular, to understand if the potential lacks of quality outlighted by QuARS are considered actually significant and noteworthy by their Requirements Engineers. The outcomes of the discussion with the Requirements Engineering have been encouraging, since QuARS has been considered a valid support for solving part of the problems they encounter during the SRS phase.

4. QuARS (Quality Analyzer of Requirements Specifications) tool

In order to make the analysis of NL SRS automatic, a prototype has been realized at I.E.I.. This prototype named QuARS (Quality Analyzer of Requirements Specifications) has been conceived to perform a parsing of the SRS phrases written in Natural Language and point out a number of potential sources of errors into the SRS themselves.



The principal functionalities provided by QuARS are listed below:

- Lexical Analysis of the SRS sentences;
- Syntactical Analysis of the SRS sentences based on a special-purpose grammar and construction of the derivation trees (structures that represents the possible derivations of the analyzed sentence by using the grammar);
- Analysis of each sentence of the SRS against the Quality Properties by using the Quality Indicators provided within the Quality Model;
- Provision to the user of warnings messages indicating those sentences having potential problems and explanation of these problems;

We don't want to provide implementation details of QuARS in this paper, but figure 1 can help understand how the prototype works.

The structure of QuARS is composed of the following key components:

- Lexical Analyser: it reads the input SRS sentences, performs a morphological analysis of the items of the sentences, verifies if they are in the English Dictionary and produces on output file the appropriate lexical category associated to each word of the sentence.
- Syntactical Analyser: it is composed of two sub-components: the Syntactical Recogniser and the Tree Builder. The first is a parser that on the basis of a special grammar and of the output of the Lexical Analyser recognise if a sentence is syntactically correct. The second component builds the derivation trees for each sentence. A derivation tree is a tree representing a possible derivation of the sentence by using rules of the grammar.
- Properties Evaluator: it is composed of two sub-components: the Evaluator of Indicators of Class A and the Evaluator of Indicators of Class B. The first performs the computation of those Indicators of the Quality Model for which information on the syntactical structure and semantic value of the items of the sentence is needed (e.g. the Underspecified Sentences Indicator). The second component performs the compilation of those indicators that don't require information on the syntactical structure and semantic value of the items of the sentence (e.g. the Vague Sentences Indicator).
- Special Dictionaries: each Indicator has a special dictionary associated that contains the special words necessary for its computation. These Dictionaries have to be considered modifiable when it is necessary to make the evaluation more suitable for particular SRS.

QuARS then has a Grammar, Special Dictionaries and the SRS Sentences as input data and provides the user with messages where the sentences having some potential problems are shown and the problems are outlighted.

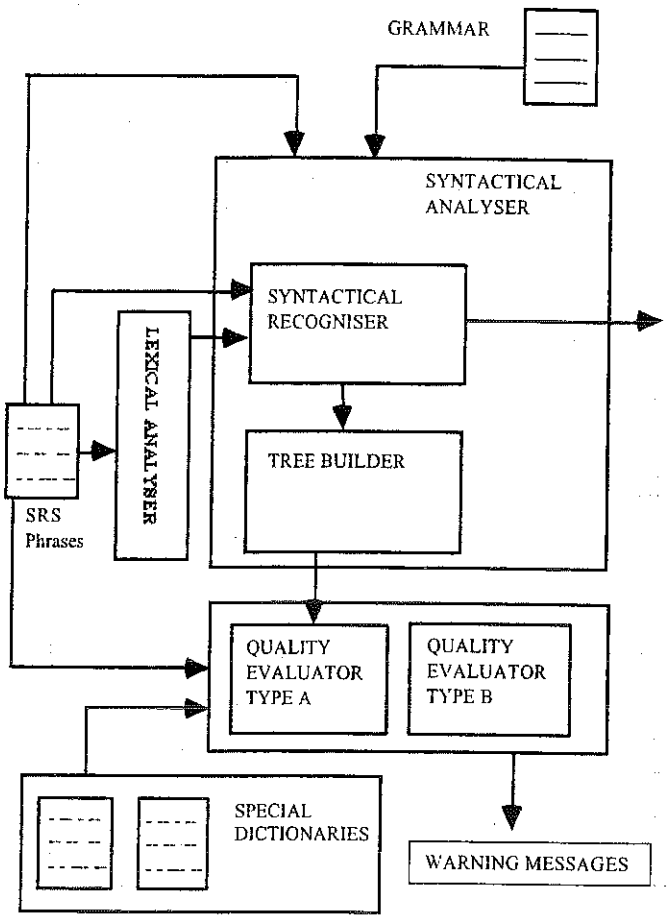
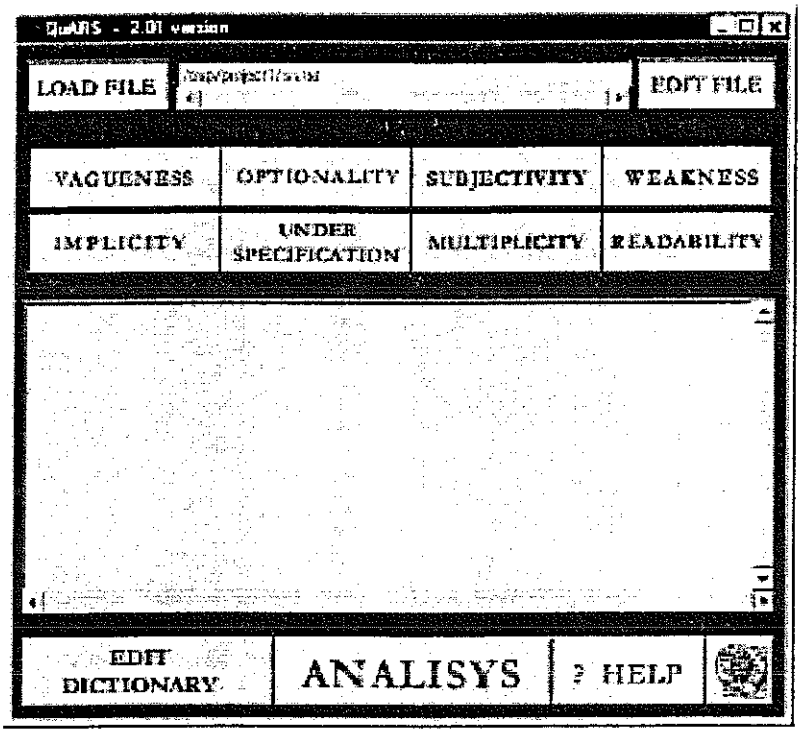


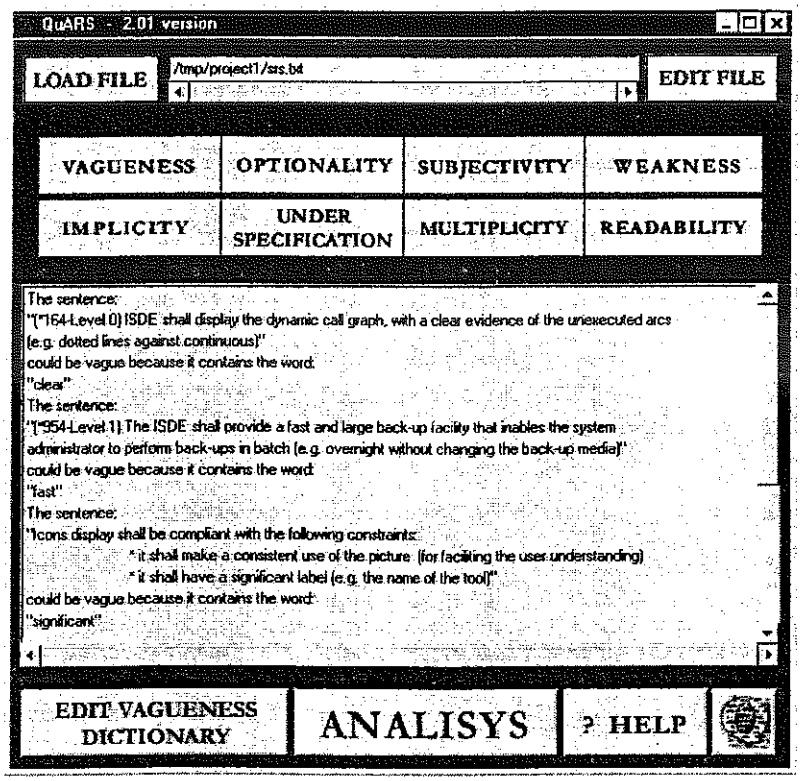
Figure 1

In the rest of this section we present the user interface of QuARS in three phases of the SRS Quality Evaluation:

- The main menu that ask against which Property we need the SRS Evaluation.



- A possible output in case of potential Vagueness problems detected.



5. The NL SRS Quality Evaluation Process

The Quality Model discussed in Section 2 drove the realization of the QuARS prototype, in fact this tool provides the real calculation of the Indicators of the Quality Model on SRS documents.
In this section we discuss how to integrate QuARS within the SRS process and in particular the SRS Evaluation process.
As mentioned in Section 1, the object of the evaluation by means of QuARS is the SRS, that, in the framework of the 'ideal' Requirements Engineering process, are precisely identified as the intermediate level between Requirements Definition and Software Specifications, or they are elsewhere simply a more or less precise definition of the tasks to be performed by the system (or by the Software component of the system). In this second case the SRS document may be the only outcome of the Requirements phase of the software life-cycle and then it may be closer to Requirements Definition or Software Design. In any case, since we aim to perform a linguistic analysis of the SRS sentences, we can, in this context, consider any document containing SRS sentences the object of the Quality Evaluation.

A typical evaluation of SRS written in NL is performed by an expert reviewer (or a team of expert reviewers) who by reading the SRS documents outlights ambiguities, inconsistencies or, in general, inaccuracies. Other actors may be involved in the preparation of these documents, but at evaluation (review) time the reviewer is the key actor.
QuARS can be helpfully included in this scheme (see figure 2). The document on which the Reviewer performs the Evaluation is analysed by QuARS that with its outputs supports the Reviewer in the detection of inaccuracies, ambiguities and linguistic inconsistencies in the document.

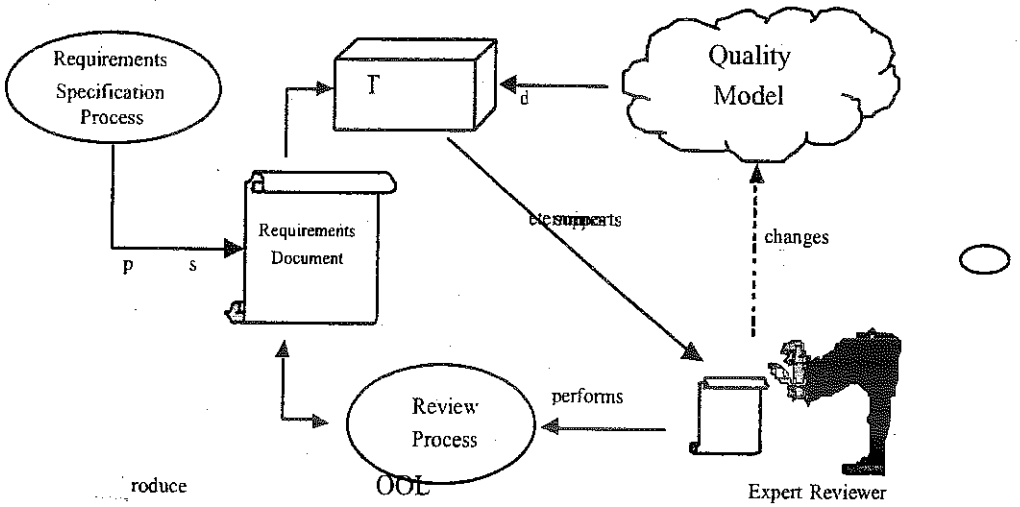


figure 2

The QuARS Tool, for its nature, can be changed in order to make it more adequate to particular context or type of SRS. Then the inclusion of the QuARS tool in the SRS Quality Evaluation process allows the reviewer to change the Quality Model if necessary in order to make it more respondent to the particular SRS under evaluation. Typically, these evolutive changes concern the contents of the special dictionaries that are the basis for the calculation of the Indicators (figure 1). The conceptual changes on the Quality Model can be easily transferred on QuARS for its modularity. Finally, the corrective actions are made on the SRS document and the process can be re-started on the modified document.

The role of the expert reviewer is then crucial in the SRS Quality Evaluation process because, not only he performs the analysis the outputs of QuARS in order to determine if the potential errors pointed out are or not concrete errors to be removed from the SRS document but he has to make also the evaluation environment suitable for the particular SRS under evaluation, by performing evolutive changes to the Quality Model and then to QuARS, if needed.

6. Conclusions and Future Work

This paper reports the work made for realizing an analyser of Software Requirements Specifications (SRS) using a Quality Model for SRS sentences based on linguistic quality properties of the Requirements. In order to automatize the application of the Quality Model to real cases, we have defined, for each Property, a set of Indicators that are tangible aspects of the requirements specifications, can be automatically calculated and provide information on the related Property.
The Quality Model has been experimented on industrial real cases of SRS in order to verify the usefulness and significance of its basis. The results of the experimentation are encouraging, since a significant number of inaccuracies and ambiguities have been detected and pointed out, and the firms that provided us with these real SRS have judged these results very interesting and useful.
Then, on the basis of this experience, we have developed a prototypal tool, named QuARS (Quality Analyzer of Requirements Specifications), that automatically performs the quality evaluation of SRS by measuring the Indicators and pointing out the sentences containing potential inaccuracies and ambiguities. The underlying idea is that avoiding these inaccuracies decreases the risk of errors in the software product due to misunderstandings or incompleteness of the SRS.

Several possible evolutions of this work may be foreseen. Engineering QuARS in order to make it a free tool for the Enterprises interested in this area will be the next step. Another evolutionary direction is to use this Quality Model and this approach to the quality of NL SRS for developing an on-line editor for the SRS. In this case the Quality Model will be used for guiding the production of SRS and not for their ex-post evaluation. Another interesting area of study can be the definition of guidelines for tailoring the Quality Model (and consequently the tool) according to particular Application Areas of the software to be developed or other criteria. These guidelines should be used by the Reviewer for the evolutionary changes to perform on the Quality Model.

7. References

[1] J. Allen: Natural Language Understanding. The Benjamin/Cummings Publishing Company, 1987.

[2] F.Fabbrini, M.Fusani, V.Gervasi, S.Gnesi, S.Ruggieri. On linguistic quality of Natural Language Requirements. In 4th 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] 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.

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

[6] 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.

[7] O.Lindland, G. Sindre, A. Sølvberg. Understanding quality in conceptual modeling. IEEE Software, 11 (2): 42-49, March 1994.

[8] 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.

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

[10] 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.

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

[12] P. Sawyer, I. Sommerville, S.Viller. Capturing the benefits of requirements engineering. IEEE Software, March/April 1999, pages 78-85.

[13] I.Sommerville *Software Engineering*, Fifth Edition, Addison-Wesley, 1995.

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