QuARS: A Tool for Analyzing Requirements

Giuseppe Lami

September 2005

TECHNICAL REPORT
CMU/SEI-2005-TR-014
ESC-TR-2005-014

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

QuARS: A Tool for Analyzing Requirements

CMU/SEI-2005-TR-014
ESC-TR-2005-014

Giuseppe Lami

September 2005

Software Engineering Measurement and Analysis Initiative

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract ..... **vii**

1 Introduction **1**
1.1 Addressing the Quality of NL Requirements: An Overview of the State of the Art 2

2 A Quality-Model-Driven Approach to NL Requirements Analysis **5**
2.1 A Quality Model for Expressiveness Analysis 6
2.2 The Linguistic Approach to Consistency and Completeness Evaluation 9

3 The QuARS Approach to Improving the Quality of NL Requirements **11**
3.1 Design of an Automatic Tool for NL Requirements Evaluation 12
3.1.1 Input and Output 14
3.2 Functional Description of QuARS 15
3.2.1 Expressiveness Analysis 15
3.2.2 Tailorability Issues 21
3.2.3 Metrics Derivation 23

4 Empirical Studies on the Effectiveness of QuARS **25**

5 Conclusions and Future Directions **29**

References/Bibliography **31**

List of Figures

Figure 1: Two-Dimensional Representation of the NL Requirements Quality 6

Figure 2: QuARS High-Level Architecture Scheme 12

Figure 3: QuARS GUI Frames 15

Figure 4: Lexical-Based Expressiveness Analysis 16

Figure 5: Active Link Between the Output and the Input Frames 17

Figure 6: Hiding the False Positives 18

Figure 7: Syntax-Based Analysis 19

Figure 8: View Derivation 20

Figure 9: Graphical Representation of a View Derivation 21

Figure 10: Creating a New Dictionary 22

Figure 11: Metrics Calculation 23

List of Tables

Table 1: Ambiguity Indicators 7

Table 2: Specification Completion Indicators 7

Table 3: Understandability Indicators 7

Table 4: Expressiveness Defect Indicators 8

Table 5: Quality Indicators vs. Linguistic Techniques 12

Abstract

Numerous tools and techniques are available for managing requirements. Many are designed to define requirements, provide configuration management, and control distribution. However, there are few automatic tools to support the quality analysis of natural language (NL) requirements. Ambiguity analysis and consistency and completeness verification are usually carried out by human reviewers who read requirements documents and look for defects. This clerical activity is boring, time consuming, and often ineffective.

This report describes a disciplined method and a related automated tool that can be used for the analysis of NL requirements documents. The tool, called the Quality Analyzer for Requirements Specifications (QuARS), makes it easier to extract structured information and metrics for detecting linguistic inaccuracies and defects.

QuARS allows requirements engineers to perform an initial parsing of requirements by automatically detecting potential linguistic defects that can cause ambiguity problems at later stages of software product development. The tool also provides support for the consistency and completeness analysis of the requirements.

1 Introduction

Creating high-quality software requirements is the first step in producing quality software. If requirements documents are incorrect, inconsistent, incomplete, or even subject to misinterpretation, there will be problems affecting both the software development and the usefulness of the end product. Methods and tools can be employed to reduce the number of potential problems. Two basic approaches exist. The language used to express the requirements can be formalized, or the procedures for analyzing the requirements can be formalized.

In recent years, many semiformal and formal languages have been developed in an attempt to reduce ambiguity, inconsistency, and incorrectness in requirements descriptions. The use of graphical object modeling languages such as the semiformal Unified Modeling Language (UML) has become very popular [Rumbaugh 99]. Formal specification languages (including the Z Notation [Spivey 92] and the B-Method [Abrial 96]) have been defined to add formality and remove ambiguities. A drawback to these languages, however, is that they are difficult for non-experts to understand, which limits their practical application. Natural language (NL) continues to be the most common way to express software requirements.

The use of NL for specifying requirements, despite its inherent ambiguity, has some advantages. NL requirements can be shared easily among various people involved in the software development process and used in several product development phases. For example, they can be used as working documents for system architecture designers, testers, and editors of user manuals; as a data source for project managers; and to establish agreement between customers and suppliers.

In the software industry, ambiguity analysis and checks for consistency and completeness in software requirements are usually performed by humans through a tedious procedure of reading requirements documents and looking for errors. Most of the errors these inspectors find are in fact simple linguistic errors. Recent market research on the potential demand for automated linguistic tools for requirements analysis concludes, “Because an engineering approach suggests the use of linguistic tools suited to the language employed in the narrative description of user requirements, we find that in a majority of cases it is necessary to use NL processing systems capable of analyzing documents in full natural language” [Mich 04].

This technical report describes a disciplined method and a related automated tool that can be used for the analysis of NL requirements documents. It aids in the creation of quality requirements by making it easier to extract structured information and metrics for detecting linguistic inaccuracies and defects.

1.1 Addressing the Quality of NL Requirements: An Overview of the State of the Art

Ambiguity in NL requirements can be reduced through the use of two types of approaches:

- The restriction of the degree of freedom in writing requirements. (This can be accomplished by using templates for structuring requirements documents or adopting a restricted English language that avoids ambiguous terms and styles.)
- Analysis of the NL text to identify and correct ambiguity defects.

Not many techniques exist for NL requirements analysis, and only a few of them are supported by automatic tools. The most popular technique for NL requirements analysis is a document inspection performed by skilled persons or teams composed of different stakeholders (e.g., developers, testers, customers, project managers). During the inspection, the person or team reads the requirements to find as many defects as possible.

The following sections summarize some recent works and studies dealing with the evaluation and enhancement of the quality of NL requirements. The studies are classified into three categories: restrictive, inductive, and analytic, each of which has advantages and drawbacks.

Restrictive Techniques

Restrictive techniques define rules that limit the level of freedom in writing requirements in NL. While they mitigate the effects of the inherent ambiguity of NL, they do not address the needs of the user as well as they do the needs of the requirements engineers because they make the requirements more precise and analyzable but less comprehensible.

In “Natural Language Processing for Requirement Specifications,” Macias and Pulman apply domain-independent Natural Language Processing (NLP) techniques to control the production of NL requirements [Macias 93]. NLP techniques are used on requirements documents to check the vocabulary used, which must be fixed and agreed upon, and the style of writing, which must follow a set of pre-determined rules set to make documents clear and simple to understand. Their technique involves the association of an ambiguity rate with sentences depending on the degree of syntactic and semantic uncertainty in the sentence, characterized by under-specifications, missing information, and unconnected statements. Their study discusses how NLP techniques can help in the design of subsets of English grammar to limit the generation of ambiguous statements.

Fuchs also describes a restrictive approach in his definition of a limited natural language, called Attempt Controlled English (ACE), which can be easily understood by anyone involved in the software development process [Fuchs 95]. ACE uses a subset of English that is simple enough to avoid ambiguities, yet allows domain specialists to define requirements in NL with the rigor of formal specification languages.

The Cooperative Requirements Engineering with Scenarios (CREWS) project aims to define a scenario-based method to elicit and validate requirements [Ben Achour 99]. This method is

based on cooperation between users and requirements engineers in defining scenarios. The scenarios express the goal in a format acceptable by both parties. The CREWS-L'Ecritoire method uses scenarios expressed in a middle ground between completely free use of NL and predefined templates. It also combines the use of informal narrative prose to express scenarios with structured NL to analyze them.

Inductive Techniques

Inductive techniques identify common problems in NL requirements and propose corrective actions or writing styles. They recommend safe writing styles for requirements but do not provide the means to apply them; hence they have a limited impact in practice.

In his paper "Writing Good Requirements," Hooks discusses a set of quality characteristics needed to produce well-defined NL requirements [Hooks 94]. He provides some common problems that arise when requirements are produced and how to avoid them. An in-depth survey is included that lists the principal sources of defects in NL requirements and the related risks.

Firesmith provides an as-yet-not-exhaustive list of the characteristics that good-quality requirements should have, along with a discussion of the typical defects in requirements [Firesmith 03].

In their research, Kamsties and Peach focus on the ambiguity evaluation of NL requirements [Kamsties 00]. They propose that ambiguity in requirements is not just a linguistic problem and suggest using checklists that cover both linguistic ambiguity and domain-specific ambiguity.

Analytic Techniques

Analytic techniques do not involve changing NL requirements, but instead use linguistic techniques to identify and remove defects. They have a high potential for impact in practice but have not been studied or developed in depth, and as a result are not very precise or effective.

Goldin and Berry have implemented a tool for the extraction of *abstractions* from NL texts, (i.e., repeated segments identifying significant concepts on the application field of the problem at hand) [Goldin 94]. Their technique is limited to a strict lexical analysis of the text.

In the paper "Automated Analysis of Requirement Specifications," Wilson and his coauthors examine the quality evaluation of NL software requirements [Wilson 97]. In their approach, a quality model is defined which consists of quality attributes and quality indicators. They have developed an automatic tool called Automated Requirement Measurement (ARM) to perform the analysis against the quality models to detect defects and collect metrics.

Mich and Garigliano propose a set of metrics for syntactic and semantic ambiguity in requirements [Mich 00]. Their approach is based on the use of information on the possible

meanings and roles of the words within sentences and on the possible interpretations of sentences. This is done using the functionalities of a tool called LOLITA (large-scale, object-based, linguistic interactor, translator, and analyzer).

Natt och Dag and his colleagues presented an approach for identifying duplicate requirement pairs [Natt och Dag 01]. Based on statistical techniques for the similarity analysis of NL requirements, this technique may be used successfully for revealing interdependencies and 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 human analysis in detecting inconsistencies and discrepancies by identifying smaller sets of requirements.

Ambriola and Gervasi aim to achieve high-quality NL requirements through CIRCE, a system that can build semiformal models almost automatically [Ambriola 97]. The system extracts information from the NL text of the requirements, then measures and checks the consistency of the models. CIRCE promotes the use of a suitable style in the requirements.

The aim of the work described in this report is to further develop analytic methods in order to provide an effective way to analyze NL requirements.

2 A Quality-Model-Driven Approach to NL Requirements Analysis

To achieve quality, NL requirements must embody several properties [Firesmith 03]. In this section, quality properties that can be addressed and evaluated through NL understanding techniques are described.

These quality properties can be grouped into three categories:

- **Expressiveness**—characteristics dealing with an incorrect understanding of the meaning of the requirements. Ambiguities and poor readability are frequent causes of expressiveness problems in requirements documents.
- **Consistency**—characteristics dealing with the presence of semantic contradictions in the NL requirements document.
- **Completeness**—characteristics dealing with the lack of necessary information within the requirements document.

The application of linguistic techniques to NL requirements allows analysis from a lexical, syntactical, or semantic point of view. "Lexical" refers to the words or vocabulary of a language, apart from its grammar or construction. "Syntax" is the way words are put together to form phrases, while "semantics" refers to the meaning of language.

For this reason it is more useful to talk about lexical non-ambiguity or semantic non-ambiguity rather than non-ambiguity in general. For instance, an NL sentence may be syntactically non-ambiguous (in the sense that only one derivation tree exists according to the syntactic rules applicable), but it may be lexically ambiguous because it contains words without unique meanings.

Figure 1 shows that the quality of NL requirements can be represented as a two-dimensional space, where the horizontal dimension is composed of the main target qualities to be achieved (expressiveness, consistency, and completeness) and the vertical dimension is composed of the different points of view from which the target qualities can be considered.

The difficulty of applying linguistic techniques varies according to the kind of analysis: in fact, while it is relatively easy to perform lexical analysis, it is much harder to deal with semantic problems in NL requirement documents. At a lexical level, only the individual words in the sentences are considered, but at the syntactical level the structure of the

sentences must be examined, taking into account the role that each word plays in the sentences. At the semantic level, the meaning of the whole sentences must be derived.

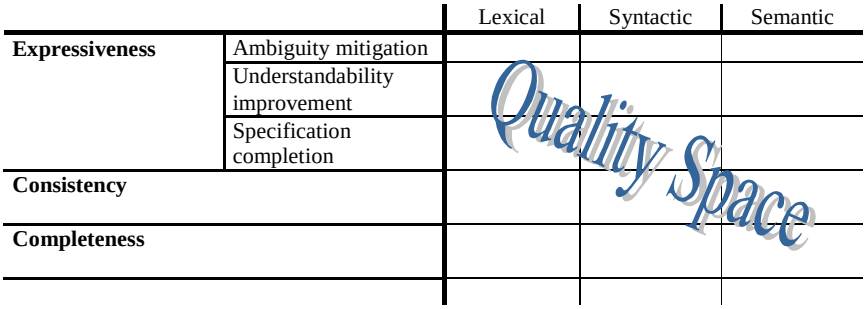

Figure 1: Two-Dimensional Representation of the NL Requirements Quality

Linguistic techniques can effectively address the issues related to the expressiveness because the lexical and syntactical levels provide means enough to obtain effective results. For this reason the quality model described in this section addresses the expressiveness property of NL requirements but does not consider the consistency and completeness properties.

2.1 A Quality Model for Expressiveness Analysis

Similar to any other evaluation process, the quality evaluation of NL software requirements must be conducted against a model. The quality model defined for the “expressiveness” property of natural language software requirements aims to provide a quantitative evaluation (i.e., allowing the collection of metrics), that helps in detecting and correcting defects, and provides the same output based on the same input in every domain.

The expressiveness quality model is composed of three quality characteristics evaluated by means of indicators. Indicators are linguistic components of the requirements that are directly detectable and measurable in the requirements document.

The expressiveness characteristics are:

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

Indicators are syntactic or structural aspects of the requirement specification documents that provide information on defects related to a particular property of the requirements themselves. Table 1, Table 2, and Table 3 describe the indicators related to each quality property and include examples of the keywords to be used for detecting potential defects in the NL requirements.

Table 1: Ambiguity Indicators

Indicator	Description
Vagueness	When parts of the sentence are inherently vague (e.g., contain words with non-unique quantifiable meanings).
Subjectivity	When sentences contain words used to express personal opinions or feelings.
Optionality	When the sentence contains an optional part (i.e., a part that can or cannot considered).
Implicity	When the subject or object of a sentence is generically expressed.
Weakness	When a sentence contains a weak verb.

Table 2: Specification Completion Indicators

Indicator	Description
Under-specification	When a sentence contains a word identifying a class of objects without a modifier specifying an instance of this class.

Table 3: Understandability Indicators

Indicator	Description
Multiplicity	When a sentence has more than one main verb or more than one subject.
Readability	The readability of sentences is measured by the Coleman-Liau Formula of readability metrics (5.89*chars/wds-0.3*sentences/(100*wds)-15.8]). The reference value of this formula for an easy-to-read technical document is 10. If the value is greater than 15, the document is difficult to read.

The defined quality model aims at identifying potential syntactic and semantic deficiencies that might cause problems when an NL requirements document is used or transformed into a more formal document. The criteria used in the quality model has been driven by results in the natural language understanding discipline, by experience in formalization of software requirements, and by an in-depth analysis of real requirements documents from industrial partners. Moreover, the definition of our quality model has benefited from the progress in the field of requirement engineering and software process assessment according to the SPICE model (ISO/IEC 15504) [SPICE 98].

The quality model, though not exhaustive, is specific enough to include a significant portion of the lexical and syntax issues of requirements documents. While it does not cover all the

possible quality aspects of software requirements, it is sufficiently specific when being applied (with the support of an automatic tool) to compare and verify the quality of requirements documents.

The sentences that are recognized as defective according to the quality model described in Table 1, Table 2, and Table 3 are not defective sentences according to the rules of the English language but are incorrect in terms of our expressiveness characteristics.

The quality model was derived to use as a starting point for the creation of an automatic tool for the analysis of NL requirements. The indicators it is composed of are terms and linguistic constructions that characterize a particular defect and are detectable by looking at the sentences of requirements documents.

Table 4 explains how the quality model indicators can be found by performing a linguistic analysis of the requirements document.

Table 4: Expressiveness Defect Indicators

Indicator	**Description**
Vagueness	The use of vague words (e.g., easy, strong, good, bad, useful, significant, adequate, recent).
Subjectivity	The use of subjective words (e.g., similar, similarly, having in mind, take into account, as [adjective] as possible).
Optionality	The use of words that convey an option (e.g., possibly, eventually, in case of, if possible, if appropriate, if needed).
Implicity	The use of sentence subjects or complements expressed by demonstrative adjectives (e.g., this, these, that, those) or pronouns (e.g., it, they). The use of terms that have the determiner expressed by a demonstrative adjective (e.g., this, these, that, those), implicit adjective (e.g., previous, next, following, last), or preposition (e.g., above, below).
Weakness	The use of weak verbs (i.e., could, might, may).
Under-specification	The use of words that need to be instantiated (i.e., flow [data flow, control flow], access [write access, remote access, authorized access], testing [functional testing, structural testing, unit testing]).
Multiplicity	The use of multiple subjects, objects, or verbs, which suggests there are actually multiple requirements.
Readability	The readability as determined by the Coleman-Liau formula.

2.2 The Linguistic Approach to Consistency and Completeness Evaluation

To find out how much support linguistic techniques could provide in the consistency and completeness analysis of NL requirements documents, we investigated tools and techniques that could be used without the adoption of formal methods.

The analysis of NL requirements for consistency and completeness begins with the identification of items that address related objects. Then it is necessary to put together all the sentences dealing with specific topic. We call such a group of sentences a *View*. Possible topics are

- quality characteristics (attributes) of the product described in the document under analysis (e.g., security, efficiency)
- components of the products described in the requirements document or belonging to the external environment (e.g., the user interface or the user)
- functionalities of the product (e.g., the printing function)

To derive a View from a document, special sets of terms are needed that represent key words related to the topic to which the View is related. We call these sets of terms "V-dictionaries."

The derivation of the Views can be made automatically using the methodology defined in this report. The methodology is based on V-dictionaries containing words that relate to a particular topic. The construction of these V-dictionaries is done by the users and relies on their skills and the study of appropriate technical documentation.

Once the V-dictionaries of interest have been built, the identification of those sentences belonging to the related View can be made automatically by relying on the output of a syntax analysis and the appropriate V-dictionary. Sentences with the subject or object expressed by terms in the V-dictionary can be tagged as belonging to that View. In other words, the idea is to put together those sentences in the requirements document that directly or indirectly deal with a particular topic.

The output of this analysis can be used by those performing the consistency and completeness analysis of a requirements document. They can then concentrate their work on subsets of the sentences in the document to make it easier to detect possible problems.

One of the principal advantages of applying the View approach is its assistance in detecting misplaced sentences. A misplaced sentence is one that deals with a particular aspect of the system that is described by the requirements but not included in the part of the document where that aspect is treated.

An example is provided in Figure 9 on page 21. The figure shows the output from analyzing the "security" characteristic in a requirements document from an industrial project. The graph represents the position of all the sentences that include the list of security words and phrases. As the figure indicates, the View comprises nine sentences. Seven of them belong to Section

3, Safety and Security Requirements, and the other two to Section 2 of the document. When the analyzers of the requirements document perform a consistency or completeness analysis of the security requirements, they will probably concentrate their attention on Section 3. Without the tool, they might miss the two sentences from Section 2.

The technique used for identifying the sentences belonging to a View is indeed a difficult and challenging one. The effectiveness and the completeness of the View strictly depend on the associated V-dictionary. The more precise and complete the V-dictionary, the more effective the outcome. Depending on the technique used and the content of the V-dictionary, some relevant terms might be missed. Some terms relating to a View might have been omitted, or the document under analysis might contain specific terms that can be understood only in context.

For example, if the View relates to the security quality characteristic of the product, a sentence like this might appear in the document: “The firewall to be used in the system is Apache.” Most likely, the term “firewall” (which is a term relating to security) will be indicated by use of the term “Apache” when it is mentioned subsequently. Nevertheless, because Apache is the name of a specific tool, we can expect that it could be not included in the V-dictionary relating to security. To solve this problem, a new methodology needs to be implemented for enriching the set of terms based on a syntactical analysis of the document. This methodology relies on the concept of subject-action-object (SAO) triplet, which can be derived easily from the output of the syntactic parser.

The idea is to select the SAO triplets which identify the action with a special verb. We might start with an ad-hoc V-dictionary, then consider, for instance, the following verb categories:

- compositional verbs (e.g. to comprise, to compose, to include)
- functional verbs (e.g. to support, to include, to use)
- positional verbs (e.g. to follow, to precede)

If the subject of the SAO belongs to the special set of terms, then the object of that SAO is also a candidate to be included in the set of terms. Conversely, if the object of the SAO belongs to the special set of terms, then the subject is also a candidate to be included. This methodology would allow for the term “Apache” to be included in the security-related V-dictionary.

To further enrich the V-Dictionaries, other related literature (such as technical documents) could be analyzed along with the requirements documents in order to identify a larger number of terms.

3 The QuARS Approach to Improving the Quality of NL Requirements

Although NL is the most-used means of expressing requirements, tools and techniques that support the analysis of these requirements are lacking. The QuARS tool, which is further described in this chapter, was designed to automate the evaluation of the expressiveness of an NL requirements document and provide support for completeness and consistency analysis.

The quality model used contains a set of indicators (derived from the linguistic elements of the sentences of NL requirements documents) that express a potential defect. Different linguistic techniques can be used in order to automatically detect these indicators within a document.

These linguistic elements can be grouped into two categories:

- lexical techniques
- syntactical techniques

The kind of analysis that can be performed using a lexical technique is a called a morphological analysis, which means it is possible to verify that the single terms occurring in the sentences are correctly written (according to the English lexicon) or suitably chosen.

The syntactical techniques are more sophisticated than the lexical techniques. Syntactical techniques rely on an understanding of the syntactic roles of each word in the sentence and of the relations among them—that is, the subject, the related verb, and the associated complements must be identified.

Some of the indicators of the quality model can be found by applying lexical techniques; others need the application of syntactical techniques. In, the correspondence between each quality model indicator and the necessary technique to be applied to make its detection automatic is shown.

Table 5: Quality Indicators vs. Linguistic Techniques

Indicator	Lexical Technique	Syntactical Technique
Vagueness	X	
Subjectivity	X	
Optionality	X	
Implicity		X
Weakness		X
Under-specification		X
Multiplicity		X
Readability	X	

The analysis of NL requirements documents should not be limited to the detection of the indicators of the reference quality model. It should also be automated for the derivation of information supporting the consistency and completeness analysis. The View derivation is a way to provide the requirements analyzer with practical support for this kind of analysis. The derivation of the Views can be made using lexical techniques, but can not be limited to the mere detection of the occurrence of terms belonging to a domain dictionary. The application of syntactical techniques is also necessary.

3.1 Design of an Automatic Tool for NL Requirements Evaluation

This section provides a high-level architectural description of the QuARS tool. It was developed to be modular, extensible, and easy to use. The architectural design provides for the first two characteristics. Figure 2 depicts the high-level architectural design.

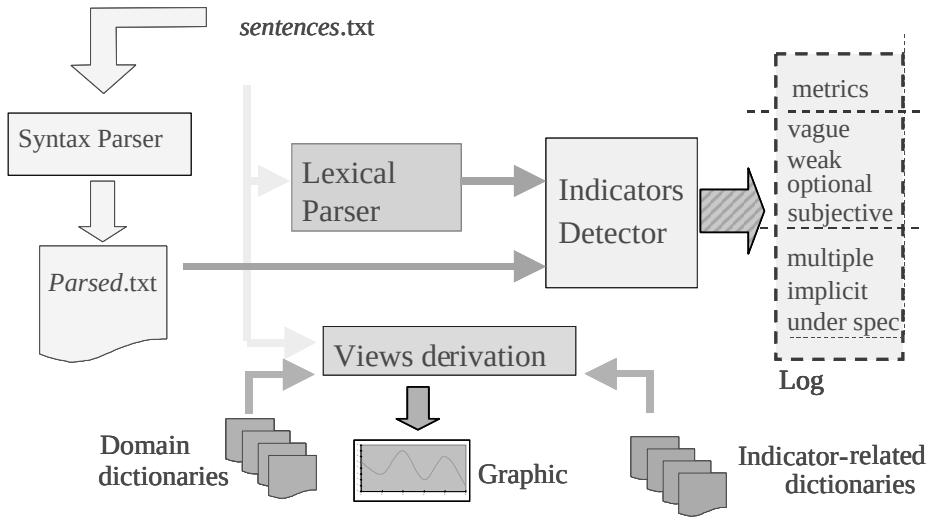

Figure 2: QuARS High-Level Architecture Scheme

The functions performed by the five main components are lexical analysis, syntax parsing, comparisons to the quality model, the construction of views, and the maintenance of dictionaries.

1. Syntax Parser—Using the Minipar application, the syntax parser derives the syntactical structure of each sentence in the requirements document[1]. Minipar associates tags with the words in the sentence to indicate the syntactical role of each of them. It also derives the relationships among the syntactical components. An example sentence follows.

   "The system shall provide the manual of the user."

   This sentence is syntactically analyzed by the syntax parser in the following way:

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

This outcome has to be interpreted as follows:

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

[1] Minipar is available at http://www.cs.umanitoba.ca/~lindek/minipar.htm.

Syntax parsers calculate one of the possible derivation trees for sentences under analysis using the rules of the English language. More than one derivation tree can exist for a sentence, and the parser might provide the wrong syntax recognition of the sentence. This problem is common to all the existing syntax parsers for English sentences. The Minipar syntax parser is one of the most widely used English language syntactical parsers and guarantees a higher rate of correctly derived sentences (about 85%).

2. Lexical Parser—The lexical parser identifies specific words and phrases appearing in the sentences of the requirements document. It is used to perform a morphological analysis of the sentences and to support analysis based on the detection of special terms or words in the requirements document.
3. Indicators Detector—The indicators detector is an original component that points out the occurrences of the indicators in the document (on the basis of the outcomes of the syntax and lexical parsers) and writes them in the log file. Along with the View deriver, this component has been fully developed using the C++ language.
4. View Deriver—The view deriver is an original component that acquires, as a first step, the structure of the document in terms of the partition of sections and subsections. Then it checks to see if a sentence belongs to a View according to the rules defined in Section 2.3. Finally, it counts the number of sentences recognized as belonging to a View occurring in each section or subsection of the requirements document. The data are represented graphically as output.
5. Dictionaries—Dictionaries are the passive components of the tool. They contain sets of terms that are necessary to perform syntactical and lexical analysis and View derivations. The number and the content of these dictionaries may vary according to the application domain and user needs.

3.1.1 Input and Output

The inputs of the tool are composed of the following elements:

- requirements document to be analyzed. The format of this input file is plain text format (e.g., .txt, .dat). This file is given to the syntax parser component, which produces a new file containing the parsed version of the sentences according to the format described on page 13.
- indicator-related dictionaries. These dictionaries may contain either the terms indicating the defects according to the quality model or the domain dictionaries to be used for the View derivation. Dictionaries must be in simple text format.

The outputs of the tool include the following:

- log files containing the indications of the sentences containing defects. A log file for each kind of analysis is produced.
- the calculation of metrics about the defect rates of the analyzed document
- the graphical representation of the Views over the whole analyzed document

3.2 Functional Description of QuARS

In this section the functional description of the tool is provided. Screenshots of the QuARS's graphical user interface (GUI) are shown to aid in the description. The GUI was developed using the TCL-TK language. The GUI when the tool is in the initial state is shown in Figure 3.

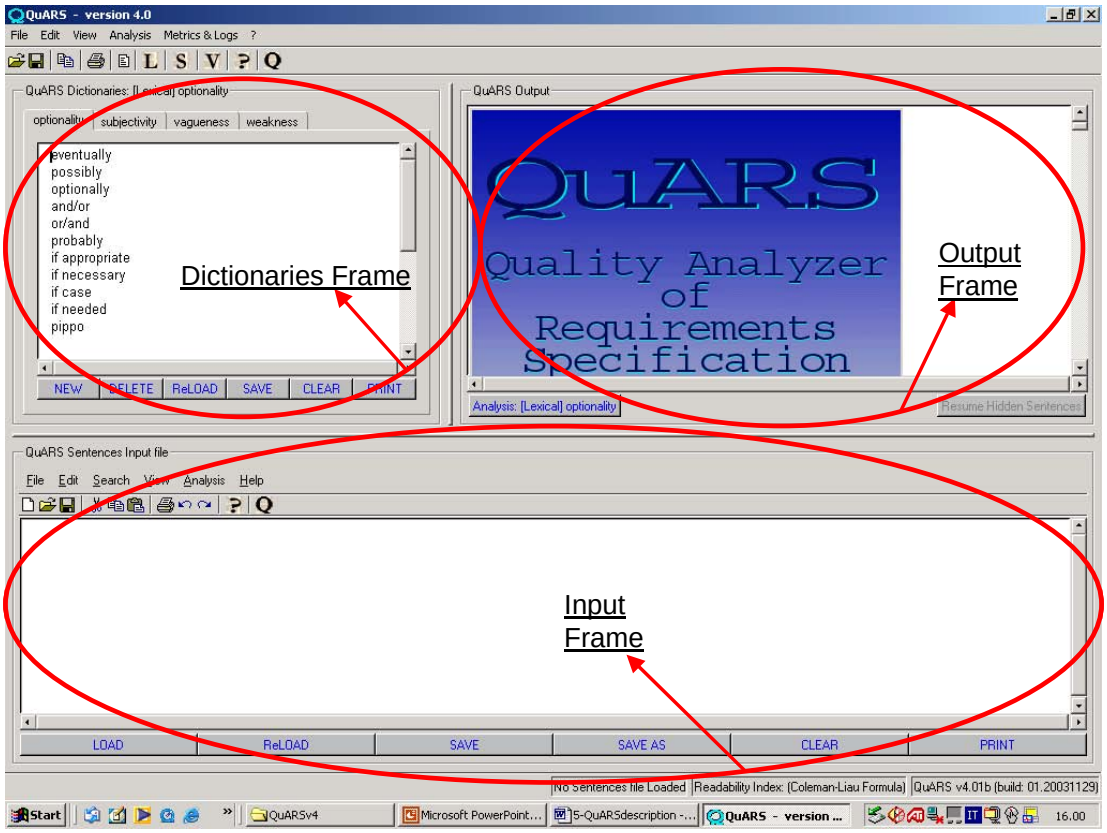

Figure 3: QuARS GUI Frames

The QuARS GUI is composed of three principal frames:

- **Dictionaries Frame**—shows the content of the dictionaries along with function buttons for dictionary-handling.
- **Input Frame**—shows the content of the text file containing the requirements to be analyzed. Function icons and buttons are provided for loading, handling, and saving the input file.
- **Output Frame**—shows the outcome of the analysis, including the requirements expressiveness analysis, support for requirements consistency and completeness analysis, and metrics derivation.

The analysis outputs are described in more detail in the following sections.

3.2.1 Expressiveness Analysis

This section describes the way QuARS performs the expressiveness analysis. The expressiveness analysis aims at detecting defects in the requirements that could cause

problems due to misinterpretation. The expressiveness analysis is conducted against the quality model described in Section 2.1. Both lexical analysis and syntax analysis are used.

3.2.1.1 Lexical-Based Analysis

As shown in Figure 4, select the "L" button on the top tool bar of QuARS (arrow 1) to perform the lexical-based analyses.

Once the lexical-based analysis has been selected, the dictionaries frame shows the dictionaries corresponding to all the available lexical-based analyses (arrow 2). Select the kind of analysis you want to perform by selecting the correspondent dictionary bookmark in the dictionaries frame. The default available dictionaries for the lexical-based expressiveness analysis are optionality, subjectivity, vagueness, and weakness.

The text file containing the requirements must be loaded before starting the analysis by selecting the LOAD button in the input frame. A window showing the options for selecting a text file from the PC file system appears. Once the input file has been selected, its content appears in the input frame.

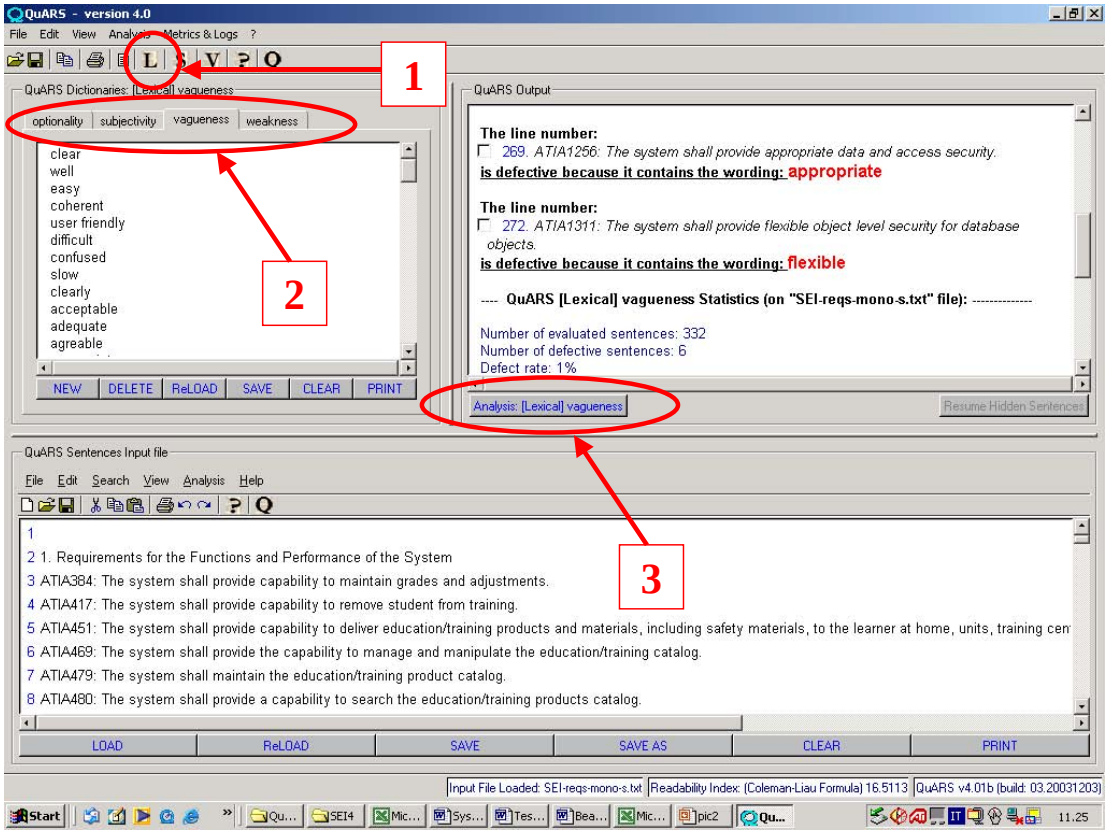

Figure 4: Lexical-Based Expressiveness Analysis

The analysis starts when the analysis button is pushed (Figure 4 – arrow 3). The output frame will show the sentences in the input file that contain the particular defect you are investigating, along with the indication of the individual term that makes the sentence

defective (according to the kind of analysis selected). Now the defective sentences can be corrected. If a sentence in the output frame is clicked, the same sentence in the input file is highlighted in the input frame (Figure 5, arrow 1). You can now correct the defective sentence in the input frame (Figure 5, arrow 2). After the all the defective sentences have been corrected and saved, the analysis can be redone to verify that no new errors were introduced.

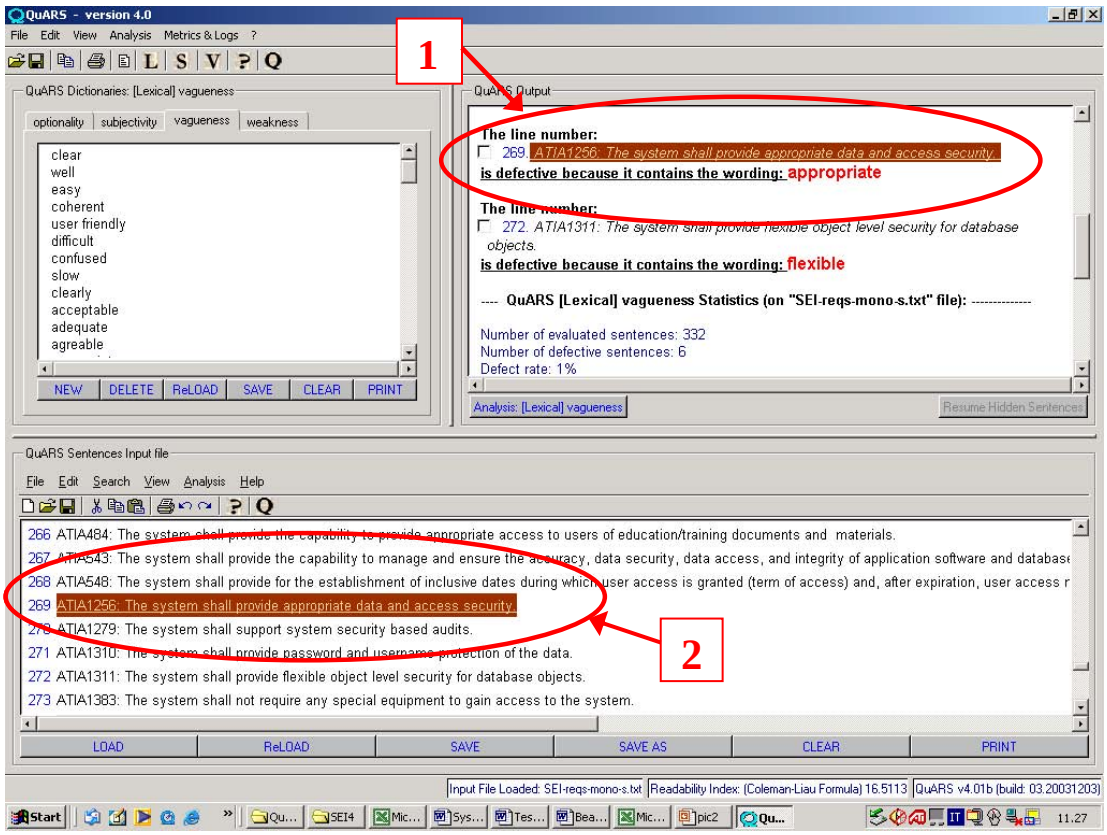

Figure 5: Active Link Between the Output and the Input Frames

The tool sometimes identifies “false positives.” A false positive is a sentence recognized as defective by the tool but considered acceptable by the user. In this case the user can activate the corresponding check boxes (Figure 6 - arrow 1). The false positive sentences are hidden in the output frame and will not be displayed even if the tool still considers them defective. The hidden sentences can be displayed again by clicking the “Resume Hidden Sentences” button in the output frame.

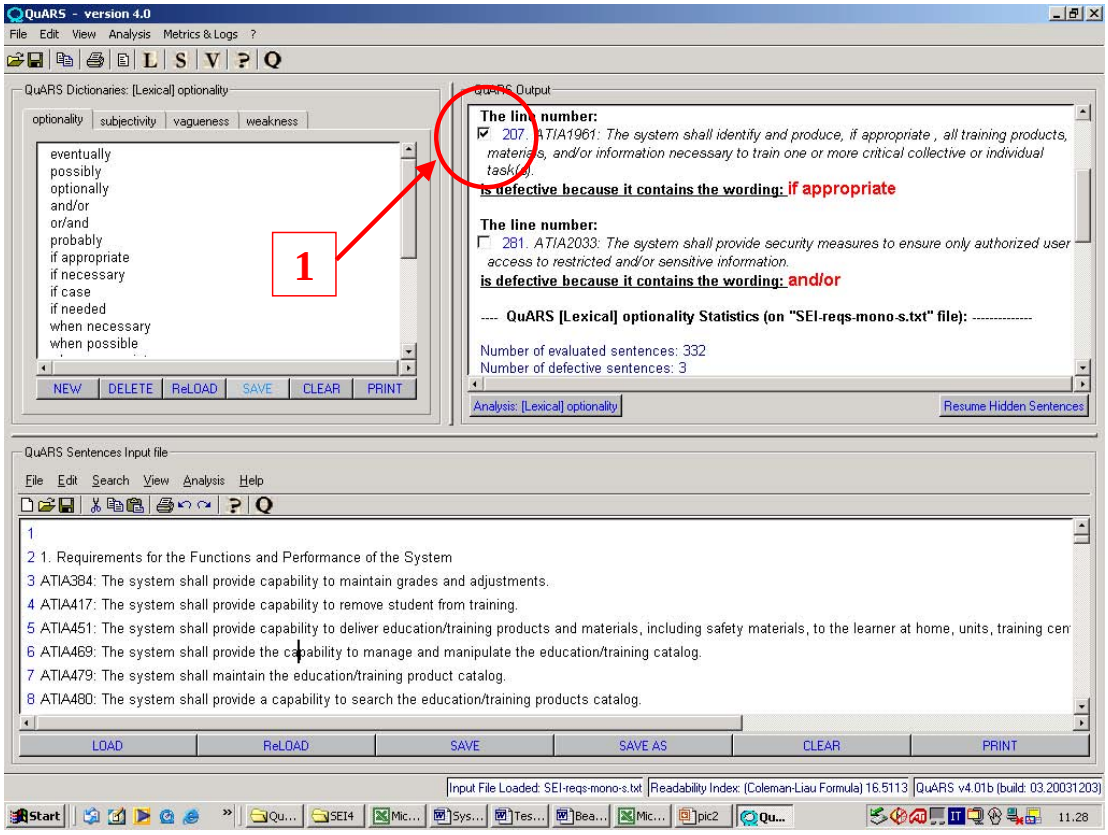

Figure 6: Hiding the False Positives

3.2.1.2 Syntax-based Analysis

As shown in Figure 7, the "S" button on the top toolbar of the GUI (arrow 1) is used for performing the syntax-based expressiveness analysis.

Once the syntax-based analysis has been selected, the dictionaries frame shows the dictionaries for each type of the available syntax-based analyses (arrow 2). Select the type of analysis from the correspondent dictionary bookmark in the dictionaries frame. The available analyses are: implicity, multiplicity, and under-specification.

The analysis is performed in the same way as the lexical analysis. For performing both analyses, the tool first derives the syntactical structure of each sentence in the input file. This structure (which is not displayed) is necessary for detecting the implicity, multiplicity, and under-specification defects in the requirements document. While dictionaries are necessary for the implicity and under-specification analyses, the multiplicity analysis doesn't need a dictionary because it relies on the syntactical structures of the sentences.

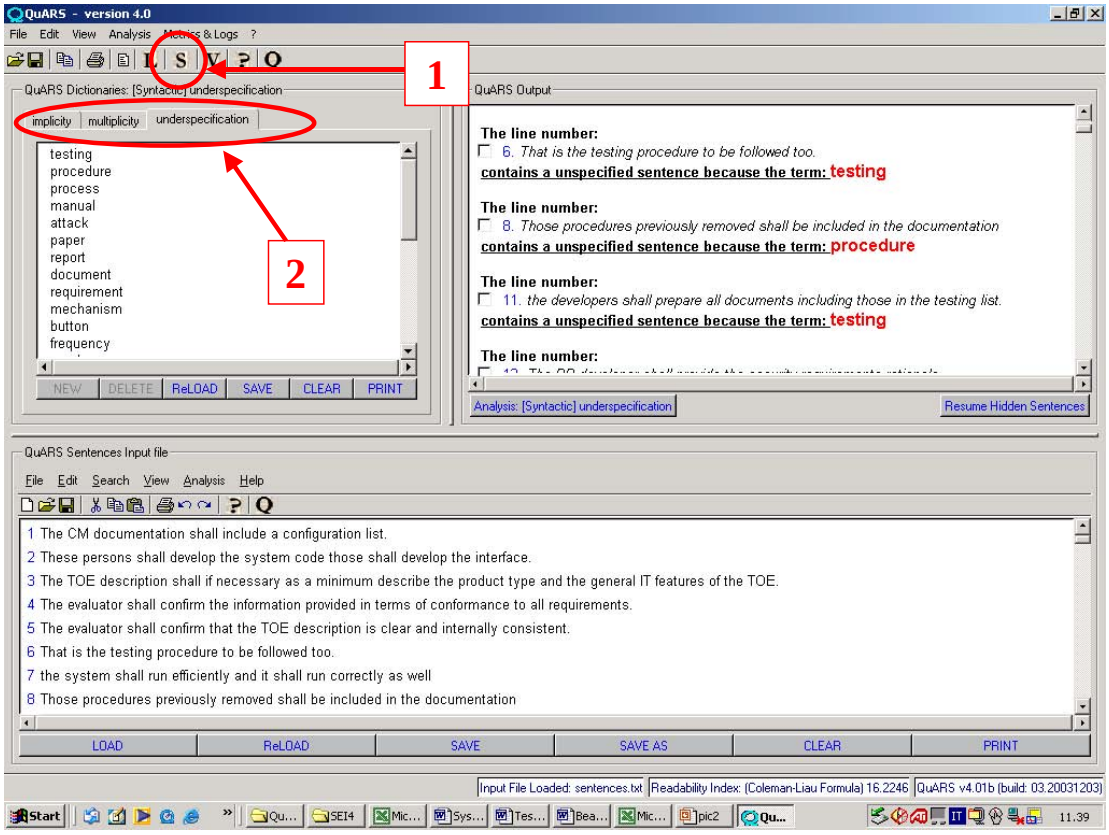

Figure 7: Syntax-Based Analysis

The QuARS tool uses the View derivation for the consistency and completeness analysis. Unlike in the expressiveness analysis, the aim is not the detection of defects at the sentence level of the requirements document, but the derivation of semantic information.

The semantic information is the “argument” a sentence deals with. A View is a filter for sentences that deal with a particular argument. The QuARS tool is able to show graphically the number of occurrences of sentences belonging to a particular View for each section of the requirements document. The View derivation relies on dictionaries, which contain V-domain-related terms instead of defect-related terms. A V-dictionary is the set of all the terms dealing with a particular View (i.e., those terms that, if contained in a sentence, indicate that the sentence is dealing with the argument to which the View is referring). The rest of this section will illustrate the way QuARS derives this information.

As shown in Figure 8, click the “V” button on the top tool-bar of the GUI (arrow 1) to select the View derivation functionality of the QuARS tool.

Once the View derivation functionality has been selected, the dictionaries frame will show the available dictionaries (Figure 8, arrow 2). Each dictionary in the dictionaries frame corresponds to a particular View that can be derived. Select the View of interest by selecting the corresponding V-dictionary.

Once the requirements file is loaded, the View derivation can be started.

The output of the View derivation is a table (arrow 3) displayed in the output frame. The rows of this table correspond to the sections or subsections of the document under analysis. The first column contains the number of sentences belonging to the View, and the second column contains the total number of sentences of the correspondent section or subsection.

These data are shown graphically by an MS Excel graph corresponding to the table.

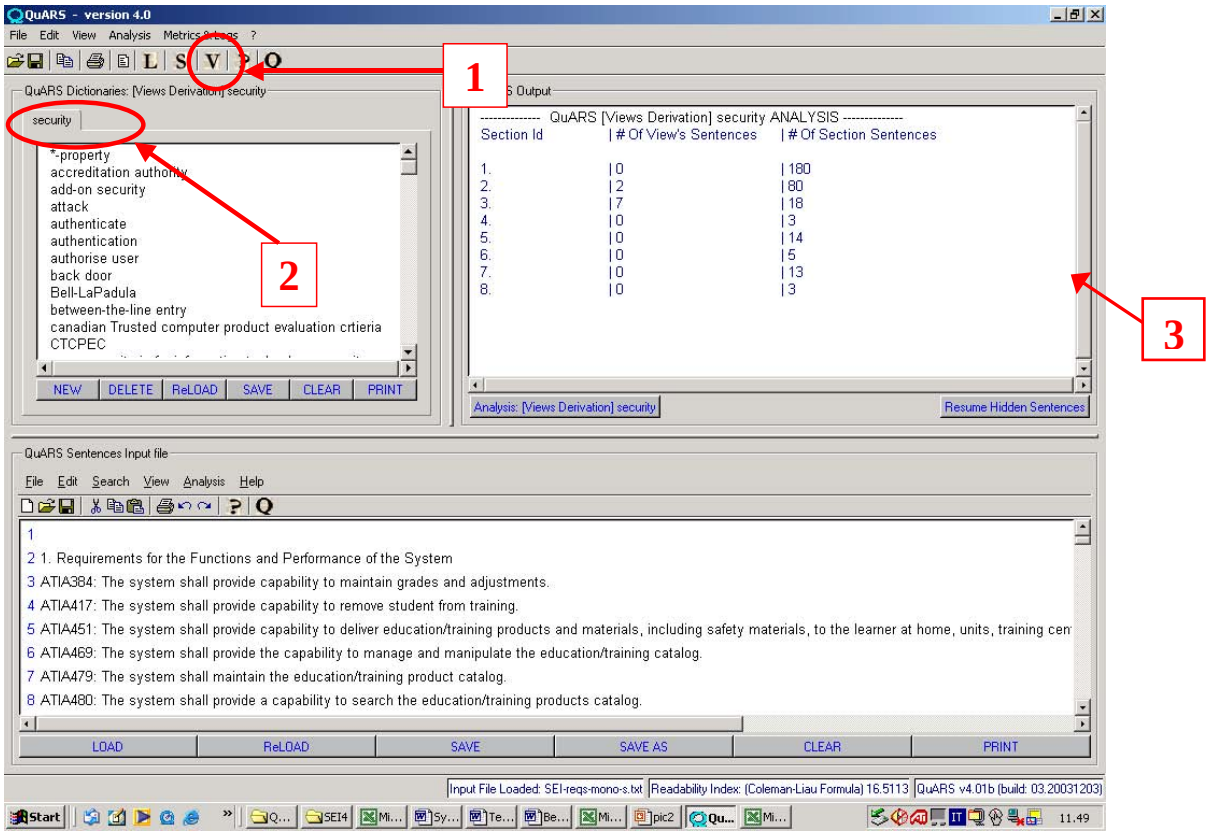

Figure 8: View Derivation

Figure 9 shows the output of the View derivation. In this case, the View derived is the security characteristic, and the dictionary in the dictionaries frame contains a set of terms directly or indirectly correlated to a security issue.

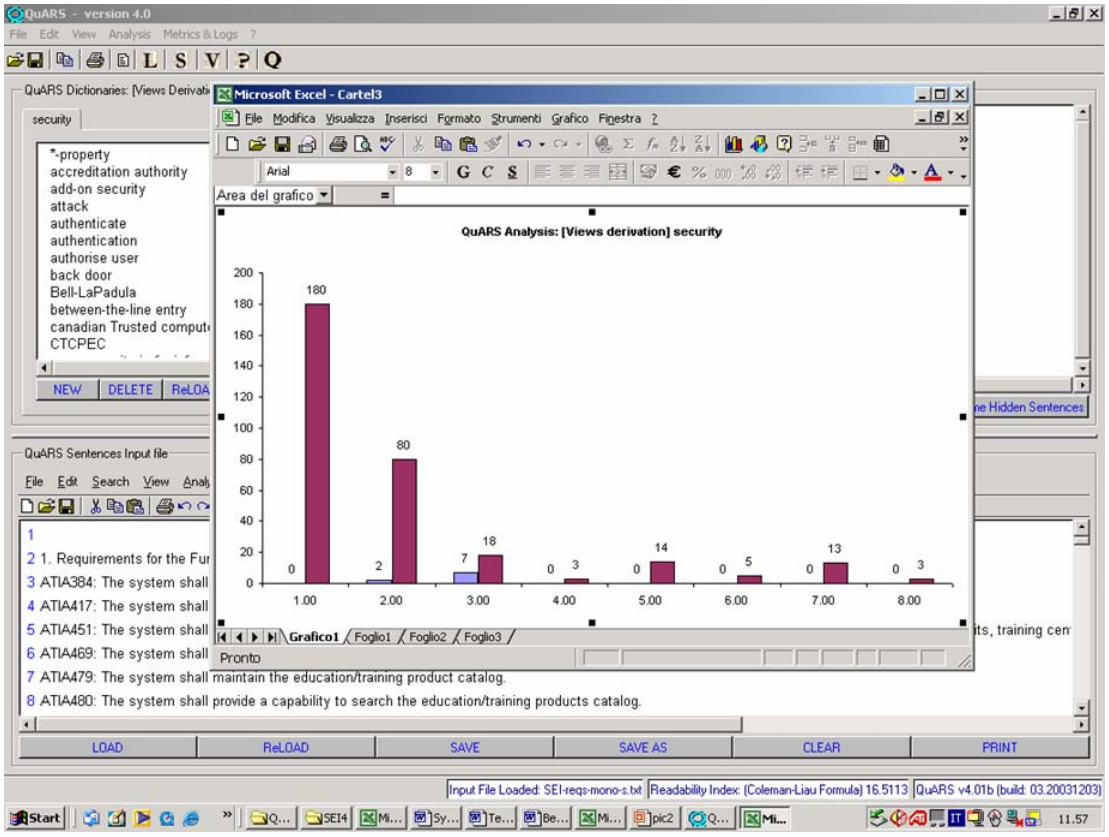

Figure 9: Graphical Representation of a View Derivation

3.2.2 Tailorability Issues

The QuARS tool was designed to be highly tailorable for different application domains and user needs. The main aspect of the QuARS tailorability is the way it handles the dictionaries (see Figure 10). In fact, it is possible to permanently modify dictionaries, create new dictionaries, and delete existing ones using the set of function buttons in the dictionaries frame (Figure 11 – arrow 1). The use of these function buttons is described below.

- **New—**used to create a dictionary. The ability to add new dictionaries for lexical analysis and View derivation allows the QuARS tool to be adapted to particular application domain and user needs. For example, for a particular application domain, it might be necessary to eliminate some terminology that could be a source of ambiguity. The creation of new dictionaries is not permitted for the syntax-based analysis because the dictionary is strictly connected to its analysis routine, and a new syntax-based analysis would require a new routine.

  A dictionary can be modified by editing it directly in the dictionaries frame. Figure 10 (arrow 2) shows the creation of a new dictionary in the QuARS.

- **Delete**—allows a dictionary to be deleted. This function is not allowed for syntax-based analysis.
- **Reload**— displays the last saved version of a dictionary.
- **Save**—makes modifications to a dictionary permanent.

- **Clear**—clears the content of a dictionary, which becomes permanent when the "save" function is selected.
- **Print**—prints the contents of the dictionary.

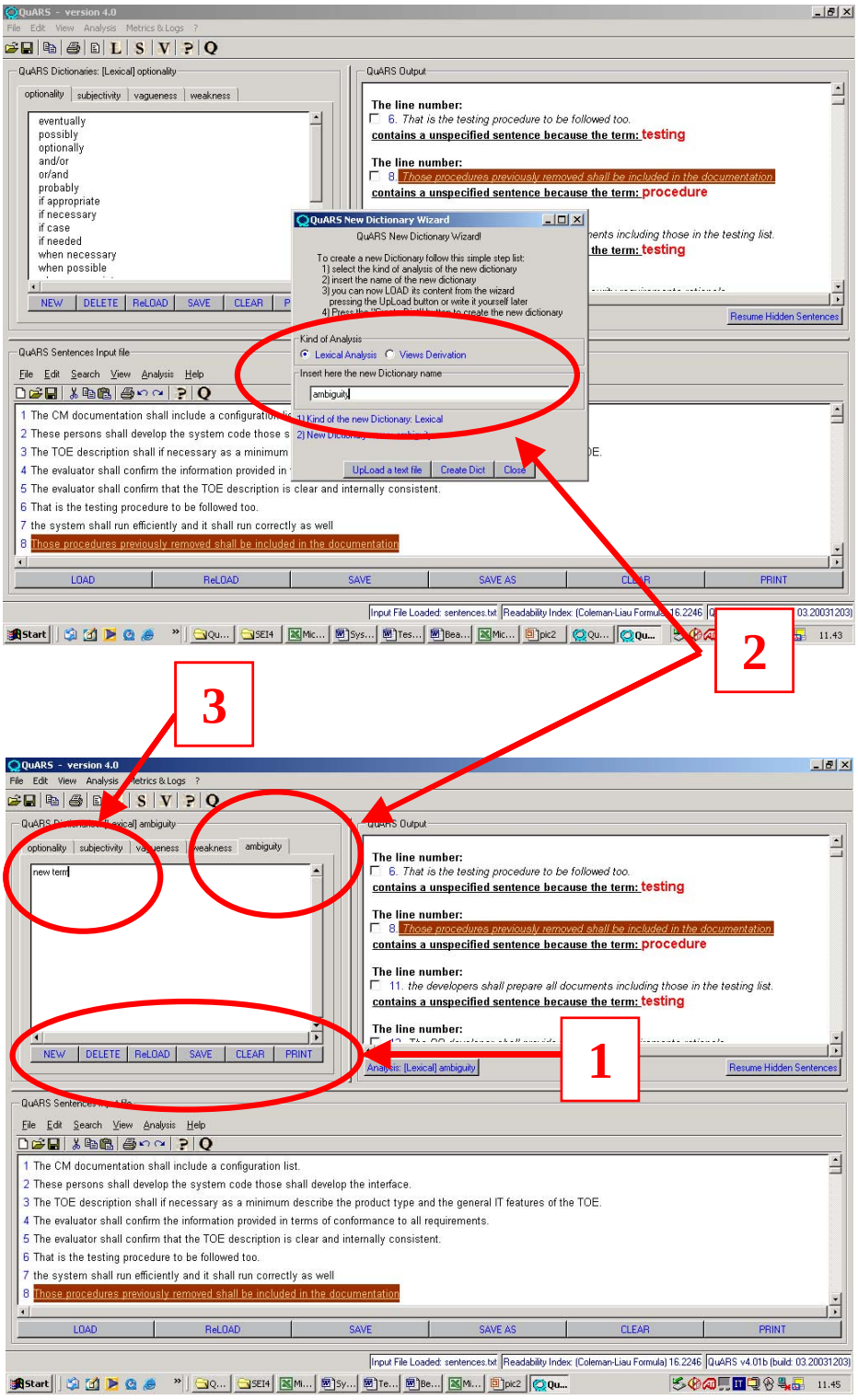

Figure 10: Creating a New Dictionary

3.2.3 Metrics Derivation

The QuARS tool can also calculate the defects rate during a requirements analysis session. Figure 11 shows how the QuARS GUI allows the user to gather these metrics.

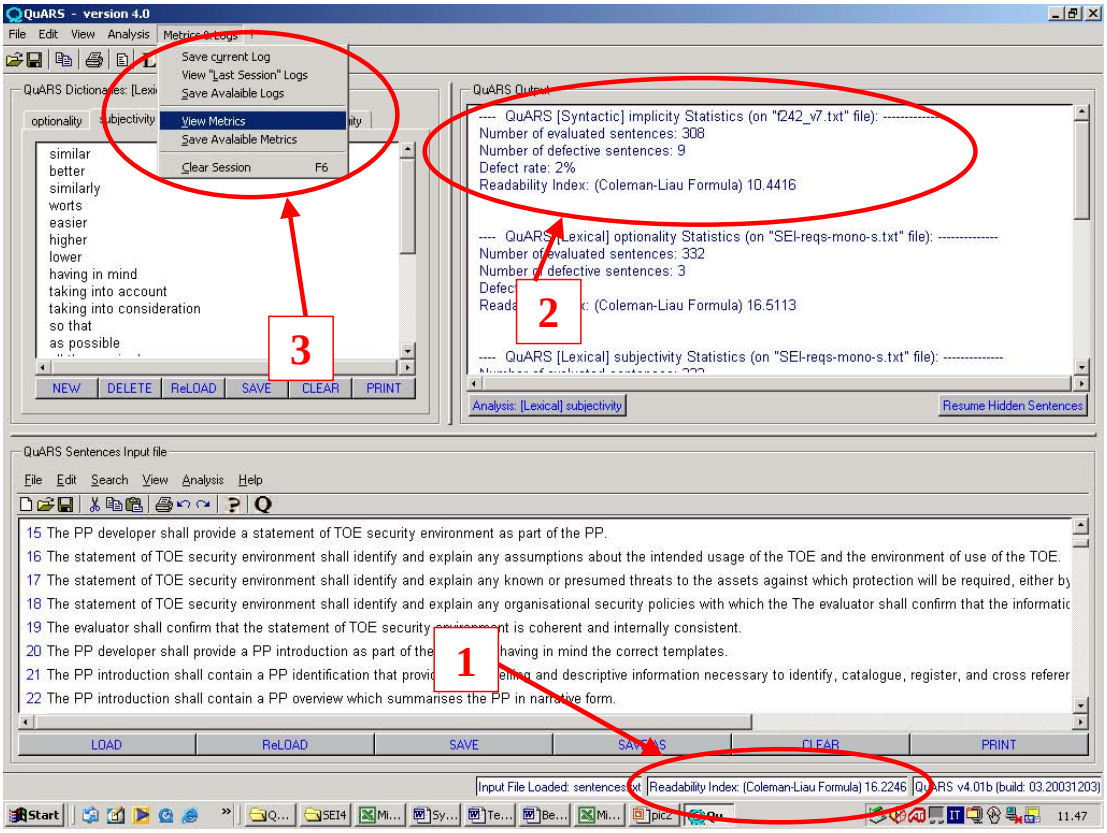

Figure 11: Metrics Calculation

Two types of metrics are available:

- **Defect rate metrics**—For each analysis (both lexical and syntax-based) the number of defective sentences is counted, and the proportion with respect to the total number of sentences in the requirements document is calculated.
- **Readability metrics**—The Coleman-Liau Readability formula is calculated (see the table on page 7 for a definition). This formula measures how easily a document can be read. The formula is also available in the bottom of the QuARS GUI (Figure 11, arrow 1).

All the metrics related to the current session can be displayed in the output frame. For each analysis performed the related metrics are maintained, even on different documents (Figure 11, arrow 2). The collection of metrics can be saved as a log file by selecting that option in the metrics & logs menu (Figure 11, arrow 3).

4 Empirical Studies on the Effectiveness of QuARS

This section describes the outcomes of two empirical studies conducted to test the effectiveness of the QuARS tool. The following initial hypotheses were tested:

1. Using QuARS reduces the time required to review or inspect requirements documents, reducing the cost of inspection.
2. Using QuARS alone or in conjunction with human inspection identifies more defects than human inspection alone.

If these hypotheses are valid, the tool will allow human inspectors to focus their attentions on understanding the business domain rather than dealing with problems in the language of the specification.

The first experiments involved testing requirements documents that had been inspected by humans. The experiments were intended to discover whether QuARS would identify further problems and to characterize the types of problems found. Two detailed cases are presented below.

Case 1: Company A, an Insurance Business

Company A uses a formal Fagan-type inspection process for requirements documents. The process has been in use for over a year. All participants had inspections experience. Company A provided us with a document that had already been inspected, but did not provide data on the cost or duration of the initial inspection. The documents were MS Word text documents and contained the following sections:

- title page
- revision control
- introduction
- item-tagged requirements statements in outline form

The document contained 445 sentences or fragments. There was little introductory or extraneous text. The outline had four levels of nesting, and the nested outline was copied and pasted into an ASCII text document.

Each of the QuARS functions was executed and the log recorded. The summary of the results follows:

- 574 sentences

- readability index (Coleman-Liau) 14.1 (college-level text)
- 234 defects found by QuARS
- 168 defective sentences, equal to 29% of the total sentences in the document. (Sentences may contain multiple defects.)

The time required to execute the procedure was approximately 60 minutes.

QuARS produced two types of false positive. The first type was the result of a specific business usage of a word that was also found in one of the QuARS lexicons. In the insurance industry, the word "commission" refers to a payment to the sales person. The default values for QuARS analysis describe this word as "under-specified." In particular, the syntax was "commission file" referring to a data file that was uniquely specified for the work at hand. "Commission" in this form represented 6 false positives.

The second type of false positive included sentences with the word "this." In the requirements, a second sentence had been appended to a specification sentence in order to make a distinction that the requirement referred to a specific use (e.g. "this" and not "that"). The antecedent for "this" would have been completely clear to either developer or tester. However, it should be noted that such sentences with implicit usage cannot stand alone. Consider the following sentence:

"Any in-house conversions or transfers into products on **this** report will show in the premium column."

This sentence cannot be properly interpreted if the prior sentence does not have an immediate antecedent for "this" naming the referenced report. Therefore, the warning about implicit usage is valuable in any circumstances where the requirements document is likely to change in ways that affect sentence order.

Processing the QuARS results by hand to eliminate the false positives was easy and took less than 1 hour. Of 234 defects, 72 were identified as false positives.

Total processing time was approximately 2 hours and 96 defective sentences were found, meaning 16% of the total sentences had escaped detection by human inspectors.

Case 2: Company B, a Manufacturer of Communications Equipment

Company B uses an independent contractor to inspect requirements. Costs and cycle time are known. The requirements are complicated by the use of both graphical and text-based specifications. Company B used the QuARS tool to analyze these documents themselves with assistance from the researchers, so we did not have access to the actual statements.

The summary of the results follows:

- 2,396 statements
- 279 human-identified defects
- 692 QuARS identified defects
- 484 defective sentences
- 179 defects reported as false positives

The human inspector was able to identify defects in graphical requirements that QuARS did not identify. On the other hand, all text-only defects identified by the human inspector were also identified by QuARS.

Company B also recorded a learning curve. The work was divided into six components. The first two components took approximately 60 minutes each to analyze. The last three components took only 15 minutes to analyze. Again, the false positives were very easy to remove. The total effort, including removal of false positives, was approximately 6 hours and the work was spread over 3 days as a part-time effort.

By comparison, the cycle time for the requirements expert was 10 days and the cost of using the consultant was approximately $6,000.

Both case studies support our second hypothesis. In both cases, QuARS uncovered significant defects that had escaped human inspectors. The cost of running the tool was trivial compared to the cost of using only human inspectors.

The first hypothesis is still not fully tested, but the evidence suggests that it will be validated. The underlying reason is complex and based on the premise that reviewers experience a maximum defect find rate. That is, if inspectors reach the threshold find rate – say 30 defects per hour – it is virtually certain they have not found all the defects they are able to identify. Companies with significant inspection experience usually require that such documents are re-inspected. The majority of requirements problems identified on first inspection are language-related problems as there are checklists and teaching methods to cover this type. By reducing the language problems, re-inspection is less likely to be needed—hence the total cycle time to process and approve requirements will be less.

5 Conclusions and Future Directions

The QuARS tool—still in prototype stage—has been used for analyzing requirements documents taken from real industrial projects to verify its functionalities and get feedback from industry about its effectiveness. The outcomes of these trials are encouraging because QuARS has been recognized as effective in terms of both number and relevance of defects detected.

The tool is highly adaptable to different application domains and requirements. It can run with almost all types of textual requirements because it uses simple text files as input, and almost all the formats produced by commercial text editors can be converted into text files.

The tool also offers many improvement opportunities because the effectiveness of the analysis performed depends on the completeness and accuracy of the dictionaries used. A method for enlarging the dictionaries and making them more accurate based on linguistic techniques has been defined and will be implemented.

Several tests are planned for the future. A high-maturity organization that develops large-scale systems intends to apply QuARS to their original requirements to compare the results generated with the orthogonal defect classification data from a completed project. The comparison will help determine whether the use of QuARS can prevent significant rework later in the product development life cycle. The results will be used for a simulation study of the cost and schedule impact of inserting QuARS into the standard process.

QuARS will also be tested in a new product development environment so effects on the process can be monitored. Such monitoring is much more time consuming, and results from this study will not be available for several months.

The use of the View dictionaries has not been fully explored. Experiments will be devised to determine how clustering by vocabulary can be used to examine requirements for omissions and conflicts.

At this time, QuARS is a research tool and not quite ready for everyday use. It currently processes only text documents and requirements by defect type. It would be more usable if the results were recorded by requirement ID so that all the problems with a requirement appear together. As a stand-alone tool, it also needs the capability to recognize a requirement ID (such as 4.3.1.a). It would be natural to embed the QuARS engine into a commercial requirements database tool.

Even as a research tool, QuARS is able to help a single individual parse over 800 requirements statements per hour and eliminate possible false positives. The few trials so far suggest that the tool is both more effective and more efficient than human inspectors. We expect, however, that the combination of QuARS with human inspectors will produce the best possible results.

References/Bibliography

URLs are valid as of the publication date of this document.

[Abrial 96] Abrial, J.R. *The B Book - Assigning Programs to Meanings*. Cambridge, England: Cambridge University Press, 1996.

[Ambriola 97] Ambriola, V. & Gervasi, V. "Processing Natural Language Requirements," *Proceedings of the 12th IEEE Conference on Automated Software Engineering (ASE '97),* Washington, D.C.: IEEE Computer Society Press, November 1997. http://portal.acm.org/citation.cfm?id=786786

[Ben Achour 99] Ben Achour, C.; Souveyet, C.; and Tawbi, M. "Bridging the Gap between Users and Requirements Engineering: The Scenario-Based Approach." *International Journal of Computer Systems Science & Engineering, Special Issue: Object-Oriented Information Systems 5,* 1 (1999).

[Cockburn 01] Cockburn, A. *Writing Effective Use Cases.* Boston: Addison-Wesley, 2001.

[Firesmith 03] Firesmith, D. "Specifying Good Requirements." *Journal of Object Technology 2*, 4 (July-August 2003).

[Fuchs 95] Fuchs, N.E. & Schwitter, R. "Specifying Logic Programs in Controlled Natural Language." *Proceedings of the Workshop on Computational Logic for Natural Language Processing,* Edinburgh, April 3-5, 1995. http:www.ics.mq.edu.au/~rolfs/papers /ifi-95.17.pdf

[Goldin 94] Goldin, L & Berry, DM. "Abstfinder, a Prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology, and Evaluation," 84-93. *Proceedings of the IEEE International Conference on Requirements Engineering,* Colorado Springs, April 1994. Los Alamitos, California: IEEE Computer Society Press, 1994.

[Hooks 94] Hooks, I. "Writing Good Requirements," 197-203. *Proceedings of the Fourth International Symposium of the NCOSE 2,* San Jose, CA, 1994. http://www.laas.fr/~kader/Hooks.pdf

[Kamsties 00] Kamsties, E. & Peach, B. “Taming Ambiguity in Natural Language Requirements.” *Proceedings of the Thirteenth International Conference on Software and Systems Engineering and Applications*, Paris, December 2000.

[Macias 93] Macias, B. & Pulman, S.G. “Natural Language Processing for Requirement Specifications.” *Safety-Critical Systems: Current Issues, Techniques and Standards*, Redmill, F. & Anderson, T. London: Chapman-Hall, 1993.

[Mich 00] Mich, L. & Garigliano, R. “Ambiguity Measures in Requirements Engineering,” 39-48. *Proceedings of the International Conference on Software - Theory and Practice, 16th IFIP World Computer Congress*, Beijing, China, August 21-25 2000, edited by Feng ,Y., Notkin, D., & Gaudel M., Beijing: Publishing House of Electronics Industry, 2000.

[Mich 04] Mich, L.; Franch, M.; & Novi Inverardi, P. “Market Research for Requirements Analysis Using Linguistic Tools,” *Requirements Engineering Journal 9*, 1 (February 2004): 40-56,

[Natt och Dag 01] Natt och Dag, J.; Regnell, B.; Carlshamre, P.; Andersson, M.; & Karlsson, J. “Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development” *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality*, Interlaken, Switzerland, June 2001. http://serg.telecom.lth.se/research/publications/docs/nattochdag_refsq2001.pdf

[Rumbaugh 99] Rumbaugh, J.; Jacobson, I.; & Booch, G. *The Unified Modeling Language Reference Manual*. Boston: Addison-Wesley, 1999.

[SPICE 98] ISO/IEC TR 15504 (Parts 1-9), 1998. http://isospice.com/standard/tr15504.htm

[Spivey 92] Spivey, J.M. *The Z Notation: A Reference Manual, 2nd ed.* London: Prentice-Hall, 1992.

[Wilson 97] Wilson W.M.; Rosenberg L.H.; and Hyatt L.E. “Automated Analysis of Requirement Specifications.” *Proceedings of the Nineteenth International Conference on Software Engineering*, Boston, May 1997. http://satc.gsfc.nasa.gov/support/ICSE_MAY97/arm/ICSE97-arm.htm

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2005	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
QuARS: A Tool for Analyzing Requirements	F19628-00-C-0003

6. AUTHOR(S)

Giuseppe Lami

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2005-TR-014

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2005-014

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Numerous tools and techniques are available for managing requirements. Many are designed to define requirements, provide configuration management, and control distribution. However, there are few automatic tools to support the quality analysis of natural language (NL) requirements. Ambiguity analysis and consistency and completeness verification are usually carried out by human reviewers who read requirements documents and look for defects. This clerical activity is boring, time consuming, and often ineffective.

This report describes a disciplined method and a related automated tool that can be used for the analysis of NL requirements documents. The tool, called the Quality Analyzer for Requirements Specifications (QuARS), makes it easier to extract structured information and metrics for detecting linguistic inaccuracies and defects.

QuARS allows requirements engineers to perform an initial parsing of requirements by automatically detecting potential linguistic defects that can cause ambiguity problems at later stages of software product development. The tool also provides support for the consistency and completeness analysis of the requirements.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Requirements Analysis, Natural Language, QuARS, Quality Analysis	43

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL