Quality Control of Software Specification Written in Natural Language

Núria Castell
Olga Slavkova
Yannick Toussaint
Antoni Tuells

Report LSI–93–50–R

QUALITY CONTROL OF SOFTWARE SPECIFICATIONS WRITTEN IN NATURAL LANGUAGE

Núria Castell[1], Olga Slavkova[1], Yannick Toussaint[2] and Antoni Tuells[1]

(1) Departamento LSI, Universitat Politècnica de Catalunya,
08028 Barcelona, Spain.
Email: castell@lsi.upc.es

(2) INRIA Lorraine & CRIN – CNRS,
54506 Vandoeuvre-les-Nancy, France.
Email: Yannick.Toussaint@loria.fr

RESUM

La creixent complexitat dels sistemes informàtics, junt amb la ràpida evolució del hardware, han posat de manifest dos problemes importants en el camp de la Enginyeria del Software: el control de la qualitat del software que es desenvolupa i la comunicació entre els participants d'un mateix projecte, ateses les dificultats que planteja l'us del llenguatge natural com mitjà d'expressió de la informació tècnica. El projecte LESD (Enginyeria Lingüística per al Desenvolupament del Software) té com objectiu el desenvolupament d'eines informàtiques que permetin: a) crear una interpretació conceptual de les especificacions funcionales o preliminars de software aeroespaial escrites en anglés, b) evaluar, usant tècniques d'Intel·ligència Artificial, la trazabilitat, la completesa, la consistència, la verificabilitat i la modificabilitat, considerats factors principals de la qualitat d'aquestes especificacions, i c) controlar la qualitat de les especificacions en base a la medició quantitativa del software.

ABSTRACT

The growing complexity of computer systems, together with the rapid evolution of hardware, has created two significant problems in Software Engineering: (1) the quality control of software, and (2) the effective communication between project participants when natural language is used to convey technical information. The aim of LESD (Linguistic Engineering for Software Development) is to develop a set of computer tools to (a) provide a conceptual representation of aerospace software functional specifications in English; (b) evaluate traceability, consistency, completeness, verifiability, and modifiability (these constituting the main quality factors) using AI techniques; and (c) control specifications quality based on software quantitative measurement.

The text contained in this report is the same as the final version of the paper accepted in the Seventh International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (May 31/ June 3, 1994, Austin, Texas, USA)

QUALITY CONTROL OF SOFTWARE SPECIFICATIONS WRITTEN IN NATURAL LANGUAGE (*)

Núria Castell[1], Olga Slavkova[1], Yannick Toussaint[2] and Antoni Tuells[1]

(1) Departamento LSI, Universitat Politècnica de Catalunya,
08028 Barcelona, Spain.
Email: castell@lsi.upc.es

(2) INRIA Lorraine & CRIN – CNRS,
54506 Vandoeuvre-les-Nancy, France.
Email: Yannick.Toussaint@loria.fr

ABSTRACT

The growing complexity of computer systems, together with the rapid evolution of hardware, has created two significant problems in Software Engineering: (1) the quality control of software, and (2) the effective communication between project participants when natural language is used to convey technical information. The LESD (Linguistic Engineering for Software Development) project links Linguistic Engineering and Software Engineering. The aim of LESD is to develop a set of computer tools to (a) provide a conceptual representation of aerospace software functional specifications in English; (b) evaluate traceability, consistency, completeness, verifiability, and modifiability (these constituting the main quality factors) using AI techniques; and (c) control specifications quality based on software quantitative measurement.

INTRODUCTION

Software project development is an increasingly complex field which often involves various teams working together. Such complexity gives rise to Software Engineering problems involving software reliability and effective communication between project participants.

Software Engineering arose out of a need to rationalize the development of software and computer systems. It covers all aspects of applied computer science, involving project management, writing code, and defining controls. The growing complexity of information systems together with the rapid development of hardware has provided the rationale for Software Engineering.

Seminal works on programming languages and design methods are well known, for instance OOD and HOOD [HOT91]. Formal languages and specification methods evidence the use of these methods at ever earlier project phases. Clearly, the earlier the mistakes are detected during project, the better. Detection of mistakes at too late stage can give rise to significant methodological changes and additional costs. Regarding costs, the correction of a mistake generated in the specification phase has been estimated [Pre87] to rise by 1.5 to 6 times when detection is made during the development phase and by a staggering 60 to 100 times when detection is made at the maintenance phase.

Natural language is most frequently employed during the development of a computer project and during its production phase. The apparent ease with which everyone uses natural language can create comprehension problems stemming from ambiguous phrasing or by inconsistencies in texts employing natural language to convey technical information. Of particular interest are the difficulties created by the use of natural language at the specification phase.

Research on natural language, carried out from linguistic, terminological or computational standpoints, is increasingly applicable to technical fields such as aerospace, nuclear engineering, telecommunications, and the like. Unfortunately works linking Linguistics Engineering and Software Engineering are rather thin on the ground. We consider that natural language processing is not only applicable to specification processing, but is also relevant to maintenance activity, software re-usability, and the production phase of a computer system [TB93].

Software Engineering norms establish the phases involved in software development and suggest software quality factors such as traceability (i.e. the ability to trace the link between the expression of a request and its

(*) This work has been partially supported by CICYT (TIC93-420) in Spain and by the Governments of France and Spain through a jointly-funded initiative ("acción integrada" HF-31B). This work was developed while the third author was a researcher in the ARAMIIHS Laboratory, MATRA-Espace / CNRS, UMR 115 du CNRS, Toulouse (France).

computer equivalent); consistency; and modifiability (i.e the ability to modify specifications easily). Formal languages do not integrate these factors or incorporate only one of them. For example, consistency is usually integrated and translated into theorem proving, modifiability is usually not taken in to consideration, and traceability is found in few formal languages. Moreover, there are factors which affect the quality of specifications and which reflect the experience of software engineers, and cannot be obtained through mathematical methods.

Using different conventions at different phases of software development makes it hard to say whether the specifications really meet the requests posed. The use of natural language at the specification stage makes it easier for project participants to reach a consensus, thus making it easier to satisfy the requests by the specifications.

Sections 2, 3 and 4 of this paper explain why the use of natural language during the specification phase is worthwhile, put forward some writing norms, and describe various systems which have already been developed. The LESD project is presented in Section 5. Section 6 deals with natural language specification processing as performed in the LESD project and with the process of constructing the requirements base. This base contains the conceptual description of all the requirements of the specifications. In sections 7 and 8, the application of artificial intelligence techniques to the evaluation of two factors involved in the quality of specifications quality is considered: traceability and completeness. The final section presents the scheme of a quality control hierarchical model based on the quantitative measurement of the quality level reached by the factors.

NATURAL LANGAGE AS A SPECIFICATION TOOL

What role does natural language play? A partial answer was given in the previous section in which the correlation between specification and request was discussed. Despite the great diversity of languages and systems which can be pressed into service during the specification phase, there are numerous situations where it is difficult to begin the design task without writing texts which offer a basic definition of the problem to be resolved and of the restrictions to be observed. Natural language allows one to combine the solutions suggested by the software engineer and client requests more effectively. Natural language is the most suitable tool for expressing the specification task given that it is an intellectual task of the software engineer.

During the initial phase, the software specifications for complex systems (such as those in the aerospace and nuclear fields) result in bulky documents since they are often written in natural language. Once created, the documentation is progressively enriched during successive phases during the industrial life-cycle of the software. Even after being formalized, the original documentation written in natural language may also serve during later phases, and during the functioning and maintenance of the developed computer system, being particularly useful for verifying the extent to which it meets designer options or client requests. Hence, software engineers continually work with the original documentation.

Natural language therefore occupies an important place in Software Engineering projects and is used in the majority of formal languages in simplified form for writing comments. Natural and formal languages are therefore doomed to co-exist, allowing one to argue about their respective capabilities and limitations.

WRITING NORMS

Documentation writing is guided by the norms which define the linguistic restrictions required to satisfy the specifications. These norms are of two types: those relating to the use of natural language in general (for example, [IEEE84] and [AECMA89]; and those that are based on terminological restrictions related to a particular domain (for example, the ESA - European Space Agency - norms). Both of them restrict the use of natural language through a set of rules which limit various irregularities (polysemy, paraphrase, ambiguity, vagueness..) which occur during the interpretation of natural language. When writing, these translate into three important recommendations: (1) the use of simple phrases (grammatically simple structures); (2) the use of restricted vocabulary (on both technical and colloquial levels); and (3) the elimination of vague expressions.

Though the norms define linguistically precise restrictions, the frequent failure to observe them makes it difficult for the consequence of such breaches to be detected afterwards. One should bear in mind that the sheer volume of documentation makes it impossible for even the most attentive reader to detect variations in the use of terminology or to check all the meanings of a phrase to discover whether another reader could interpret it in a different way.

In addition to linguistic restrictions, the norms also include Software Engineering constraints related to the quality factors of the specifications, such as consistency, completeness, traceability, modifiability, and verifiability.

PROCESSING THE NATURAL LANGUAGE SPECIFICATIONS

As far as natural language processing for software specifications is concerned, there are two different (but not mutually-exclusive) approaches. The first one emphasises only the superficial characteristics of the texts written in natural language (words and groups of words) and its purpose is to manage a set of requirements. The second one is concerned with information content, i.e. semantics.

The first approach is based on the development of environments, mainly beginning from database management systems, which aid the engineer in the tasks involved in classifying and storing the specifications. These systems allow automatic insertion of a traceability table at the end of documents. However requirement-associated relations are "hand-made" by the engineer when

the requirement is introduced. The ARTS [DF84] or SWIFT systems provide examples of this approach.

The second approach tackles the use of specifications in natural language by taking into account the experience of research on natural language processing, in particular the use of its logic-semantic structures. This approach implies the need to deal with semantic representations. The construction of the semantic representation of a set of requirements in a particular domain requires not only linguistic knowledge (i.e. of syntax and semantics) of the language the requirements are written in, but also in-depth knowledge of the domain. We favour careful study of natural language as specification tool and hence adhere to this second approach. We shall briefly comment below on some of the work based on this approach.

Let us first consider the SEC system (Simplified English Checker). This is an automatic verifier of text style, developed by Boeing to help the writers of aerospace maintenance documentation adapt their work to writing norm criteria, such as AECMA PSC-85-16598. A description of this system can be found in [WHH90]. The SEC system provides the writer with a report which states where and why the revised documentation has deviated from the norm. In 1990, this software correctly processed 80% of the introduced text, and the company calculated that it produced a saving of 12,000 to 16,000 working hours per year. Besides raising the quality of the written documentation, the use of SEC has an educational spin-off since writers learn to correct their most frequently made mistakes and reduce the frequency with which they crop up in future texts.

Secondly, one should also mention another work which, while more original, we consider to be less realistic. The system is described in [SNM*88] and aims to translate specifications written in natural language into formal language. The formal language for the specifications consists of an algebraic language based on a context-free grammar and a set of axioms defining the semantics of the language. This approach appears to be unduly restrictive since the parser used is very small and does not address the problem (among others) of quantification, without which it is impossible to verify consistency at the formal language level. Moreover, automatic translation does not allow the incorporation of essential information implied in the natural language specifications into formal expressions. In conclusion, we consider writing specifications in such a restricted natural language is tantamount to writing out algebraic expressions using quasi-natural language phrases.

Thirdly, one should also mention the work done by Carver and Cordes ([CC88], [CC89], [CCG89], [Cor89]). Their method of evaluating the quality of the specifications written in natural language is a two-stage process: (1) text parsing in order to obtain a formalized specification environment; and (2) an analysis of consistency, completeness, and traceability, based on the environment obtained in the previous phase. At the end the system provides a report on the analysis performed. To construct the formalized environment, the text is parsed phrase by phrase (the parsing is, in, fact, syntactical) and then a facts base is created, coded in Prolog. Different algorithms are applied to this facts base in order to perform the above-mentioned quality controls. This work defines a methodology rather than develops a system.

THE LESD PROJECT

The LESD project (Linguistic Engineering for Software Development) [BTB91] was instigated by the ARAMIIHS center in Toulouse (France). This center was established through a joint agreement between the CNRS and MATRA MARCONI SPACE company. The project is carried out by researchers from IRIT (Institute de Recherche en Informatique de Toulouse), from the Paul Sabatier University of Sciences, from Le Mirail University of Humanities, from the MATRA company, and, since 1991, by several researchers from the Technical University of Catalonia through a Spanish-French grant-funded joint initiative.

LESD aims [TBB*91] to develop computational tools which will (1) allow conceptual interpretation of functional or preliminary software aerospace specifications written in English; (2) permit evaluation of quality factors (see Section 3 above) by means of reasoning algorithms applied to the conceptual representation; and (3) help the engineers handle documentation. The long-term LESD aim is to develop a writing assistance system which would enable one to identify problems related to the use of unidentified terminology and complex or ambiguous grammatical forms.

This project involves an in-depth study of the language and, thus, integrates lexical, grammatical, and semantic components. In addition the domain knowledge has been considered and represented as it is vital to conceptual interpretation and the reasoning process ([BBC*92], [Tou92]).

THE LESD ARCHITECTURE

The present LESD architecture (see figure 1) consists of two parts. The first part concerns syntactic-semantic and domain analysis of requirements, to obtain their conceptual representation (this part of the work thus belongs to the Linguistics Engineering field). The second part is related to Artificial Intelligence and embraces reasoning mechanisms applied to the representation of the requirements.

Requirements Analysis

The requirements, from which the specification is drawn up, consist on two or three short sentences written using restricted vocabulary and syntax, as usual for documents belonging to any technical domain. In few cases conjunctions or disjunctions appear and we have defined a possible representation for this kind of sentences. Also some aspects of the quantification problem have been considered. At present, the requirements are processed sentence by sentence, leaving out the reference problem. However this problem is easier to treat in our case than in a general natural language processing.

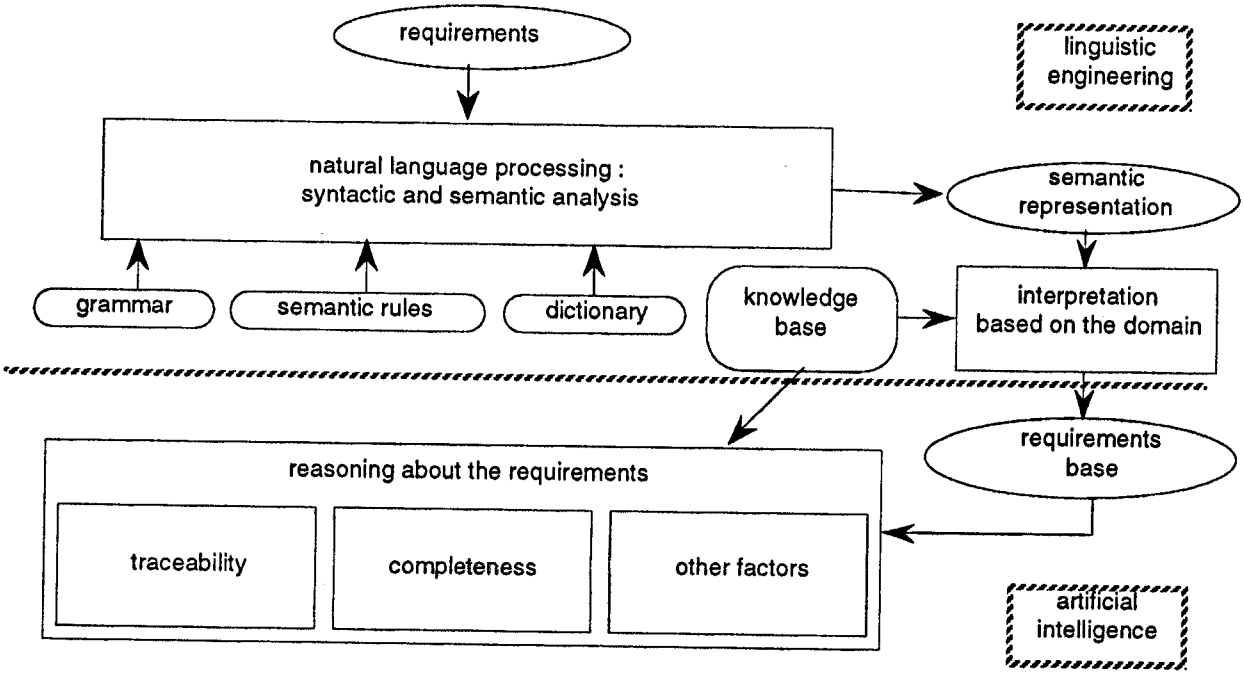

FIGURE 1 The LESD Architecture

The requirements are successively analyzed using the ALVEY parser [BGB*87]. This parser has been adapted to our domain. After semantic and functional analysis, each requirement is interpreted taking into account the domain representation. The generated representation is then incorporated into the requirements base.

Although the parsers have not yet been fully-developed, substantial work has already been done on the definition and implementation of the knowledge and requirements bases. A typology of domain objects and activities, including their relationships, has also been defined. This was implemented using frame-based formalism.

To illustrate the process involved, consider the following real-life requirements:

req-1: The IOI-GS shall monitor the systems of the space vehicle

req-2: The IOI-GS shall control the automatic systems of the space vehicle

On completion of the requirements analysis, the conceptual representation of requirements is integrated step-wise into the requirements base, as shown in figure 2 [Tou92]. The first level of the figure shows the part of the knowledge base needed for interpreting the requirements given above. The other levels show the step-wise construction of the requirements base. During this process, concepts may be learned and integrated into the knowledge base, for instance the concept of "automatic systems of the space vehicle".

Requirements reasoning

The reasoning process concerning requirements is divided into different modules, each one corresponding to one quality factor. The reasoning mechanisms use the requirements representation contained in the requirements base, and the domain knowledge base. Thus far we have paid particular attention to two quality factors which are especially relevant to software design and which we shall expand upon in the sections below.

TRACEABILITY

Working from the client's requests, the software engineer writes the specifications of the system and subsystems in top-down fashion. This initial definition phase is followed by other phases in which different modules are developed, being integrated in bottom-up fashion. Traceability expresses the links between the requirements at different levels. It is useful to know the origin of the requirements during the top-down process. During the bottom-up process, traceability facilitates the control tests and allows one to identify the initial requirements which remain unsatisfied.

There are different approaches to the design of tools aiding the engineer in traceability tasks. When specifications are written in a formal language, traceability control is based on their algebraic properties. Another approach is based on the use of a database management system and the construction of a specifications development environment. In these systems traceability links are "hand-made" by the engineer. More recent approaches provide for the use of natural language, as for instance, the system described in [SNM*88] which is based on the automatic translation of natural language into formal language.

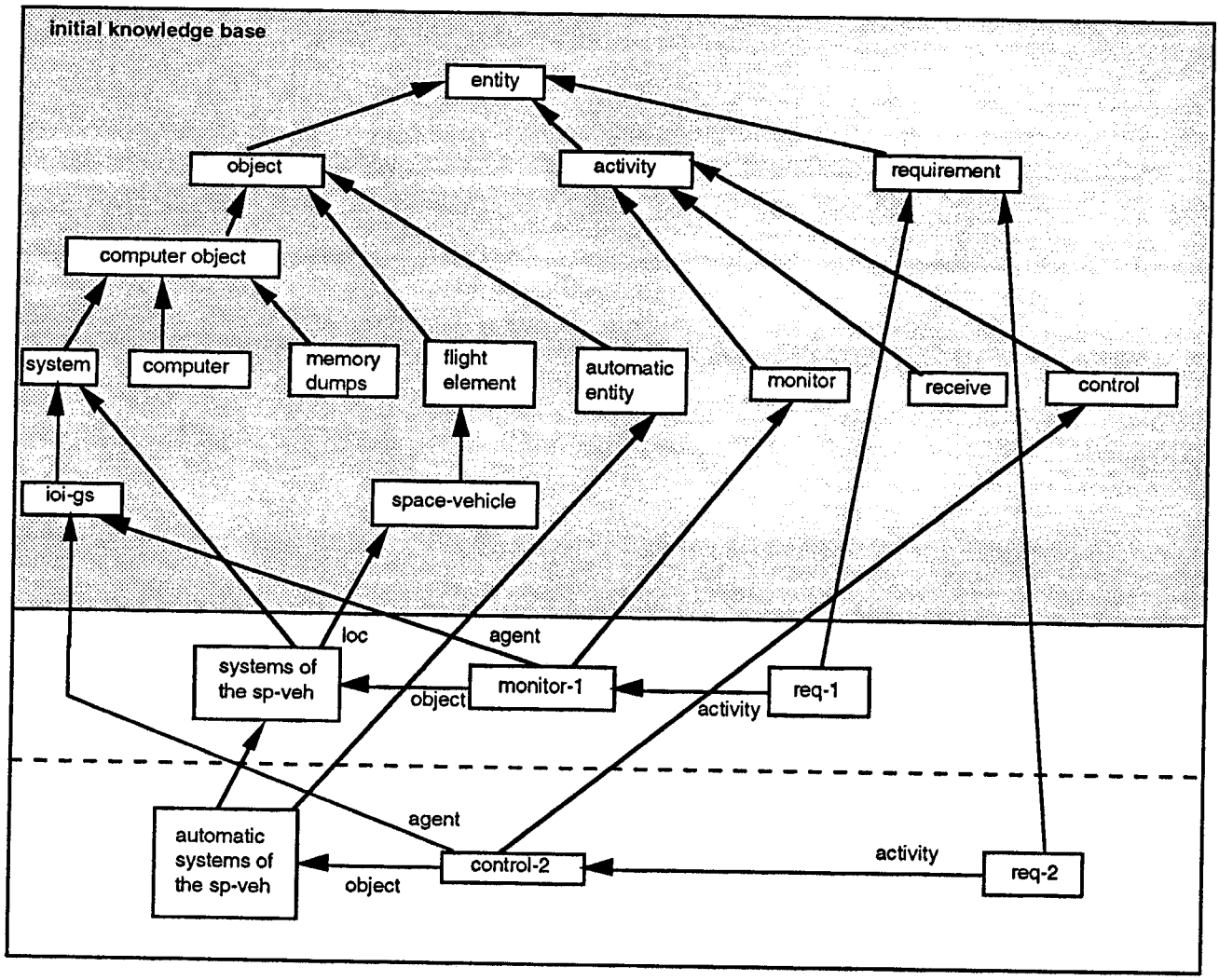

FIGURE 2 Diagram showing the process of incorporating requirements. Arrows without label stand for "is-a" relations.

Traceability in the LESD project

The LESD project posed the question as to what linguistic criteria (semantic or pragmatic) are used to create traceability links between requirements.

Traceability control does not imply the simple finding of links between all the concepts involved in any two requirements. Rather, this control implies the use of inference techniques in a well-structured knowledge representation of the universe of the discourse. One of the initial goals of LESD was to define a set of relations to structure the lexicon and the domain concepts. We have created taxonomic relations (the "is-a" relation), meronomic relations (a breakdown of the object into components), temporal relations (especially between activities), characterising relations (for instance, "status" characterises "system") and thematic functionality relations (as "agent", "object", etc.).

At present, there is no formal definition of traceability which could allow implementation of an algorithm that automatically generates traceability links. That is why we have chosen an interactive approach where the engineer poses a question by identifying a set of concepts interconnected by relationships. The answer of the system is a list of requirements, which conceptual representation contains these concepts and relationships. The developed algorithm, for the analysis of questions and the generation of answers, is based on the idea of type. Thus, the selected requirements are the ones whose concepts belong to the concept type given in the question. This allows one to classify the requirements from the most specific to the most general ones according to the derivation (by means of an "is-a" relation) between the concepts given in the question and the concepts contained in each requirement. The generation of answers implies activating the inference mechanism in the knowledge base.

The questions can be very simple (for example, "Get all the requirements related to one type of object") or very much complex, involving activating the inference mechanism in the structure and content of the knowledge base. For instance, "Get all the requirements which express data monitoring" will include the following requirements in the answer:

req-1: The system shall monitor the data of the space vehicle.

req-2: The IOI-GS shall receive the data of the space vehicle.

The system can relate these requirements using the taxonomy of objects and their relationships. For example, "IOI-GS" is a sub-class of "system", the "receive" action is a sub-action of "monitoring", and "data of space vehicle" is a sub-class of "data". Moreover, the system detects the fact that the concepts involved in requirement 2 are more specific than the concepts in requirement 1. Hence, satisfying the second requirement provides only partial satisfaction of the first requirement.

COMPLETENESS

What does completeness mean in the context of software specifications written in natural language? The word "completeness" is used in so many scientific fields that it is sometimes difficult to say whether it always has the same meaning. This point is illustrated below with specific examples, particularly with reference to the concept's relation to natural language processing. For a more detailed description see [TC93].

Deductive Completeness

This notion of completeness is used in any formal system. It is, for example, the one used in Gödel's theorems of incompleteness. A possible definition might run as follows: "a formal system is complete if for any proposition A of the system, A or (not A) can be deduced".

Expressive Completeness

Expressive completeness is the concept related to the expressiveness (or lack of) of the programming languages, linguistic formalisms or knowledge representation systems. Obviously, the lack of this expressiveness depends on what we want to express, i.e. on the ability to represent the knowledge types required by the domain.

Structural Completeness

We speak about structural completeness when the existence or non-existence of a sub-structure within a more general structure is involved. There are many examples of these types of structures. The most famous of them in natural language processing are Schank's scripts [SA77], and task- oriented dialogues [Gro86]. It is worth mentioning that these structures are more or less complete with respect to the world modelled (for example, they typically implement a user model).

As far as Software Engineering is concerned, solutions have been suggested (similar to those mentioned above) to control the completeness of a set of specifications. For instance, "The Requirements Apprentice" project [RW91] uses the concept of "cliché" (which is like a frame), related to a particular domain. In each cliché some slots have default values while other ones require the introduction of a value. A requirement is not complete if any of the slots for which a value has to be entered remain unfilled.

Conceptual Completeness

This is related to communication between various persons and thus to natural language, which is the most commonly used form of communication. From this point of view, a communication act (which for the sake of simplicity, is taken here to mean the transmission of a message from the sender and its reception by the receiver) is complete if the receiver understands the information which the sender is trying to communicate. As far as software specifications written in natural language are concerned, we can say that they are complete if they allow the software engineer to understand the function of the system. The main problem involved in these specifications (and which makes them worthy of study) is that they are written by one person and read by another.

The Control of Incomplete Specifications in the LESD Project.

For this control it is necessary to construct an actions - sub-actions hierarchy, representative enough to guarantee its appearance in any set of specifications related to an aerospace domain. For example, in our knowledge base the action "monitoring" is broken down into three sub-actions: "receive", "analyze" and "visualize".

Consider the following real-life example of a requirement: "During the launch phase, the IOI-GS shall analyze and display the status of the space vehicle".

If, after analyzing this requirement, the system observes that there is no reference to "monitoring" action, it will alert the engineer that a requirement relating to this action may be missing. Likewise, if after the analysis of the requirements, the system notes that one requirement makes reference to "monitoring" action but none makes reference to its descendants in the hierarchy, it can then prompt the engineer by indicating that the specifications are incomplete.

This approach is based on reasoning mechanisms applied to the knowledge base. It is vital that this base be reliable and precise. The engineer must control the consistency of the inferences executed by the system, and validate what requirements are necessary. Moreover, the engineer has to reach a decision on the completeness of the specifications. This decision is not automatic because our possible solution – as well as that suggested by [RW91] – tries to "capture" the conceptual completeness of the specifications through structural completeness of some structure (in our case, of action - sub-action hierarchy), i.e. by merging two types of completeness. This approach suggests a limited solution to the completeness problem on which we are working.

QUALITY MEASUREMENT

The specification phase clearly influences software quality. This is why it is essential to rigorously control specification quality, evaluating each factor involved. Evaluation of the quality factors in specifications, from the LESD standpoint, implies the development of reasoning algorithms applied to the conceptual representation of the functional or preliminary specifications written in natural language. The algorithms developed allow one to formally define the breakdown of factors to stablish quantitative criteria of the quality of

these factors. The quality of these criteria is much easier to evaluate than is the case with the factors. Evaluation of the quality relating to each criterion is performed by software metrics. Metrics are procedures which define the quality evaluation of the above-mentioned criteria according to the direct measures of their component elements (based on the corresponding algorithm). Quantitative measurements of the component elements' quality are functions with these components as parameters, producing a number which is interpreted by the corresponding metric to provide a quality evaluation. The formalism in the definition of factors, criteria, quality components and all their inter-relationships guarantees objective and rigorous control of specifications.

In general, the software quality control model in LESD exhibits the following hierarchical structure [Sla92].

1st. level: reflects the principal aim of the quality control system, which in the LESD case represents the quality of specifications.

2nd. level: is defined by factors which, in our case, are: traceability, consistency, completeness, verifiability, and modifiability.

3rd. level: is defined by the quality criteria. For example, the traceability factor [BBC*92] is broken down into the following criteria:

1-traceability of all the requirements;

2-links between the requirements at different integration levels of the modules.

4th. level: corresponds to software metrics. The metrics corresponding to the two traceability criteria may be defined as follows:

1– relation between the number of traceable requirements and the total number of requirements;

2– relation between linked requirements at different module integration levels and the total number of links between these levels in accordance with client requests.

5th. level: corresponds to measurement (quantitative measurements) of quality components. To calculate the metrics mentioned above, the measures of the following quality components are required:

1– the number of traceable requirements;

2– the total number of requirements;

3– the number of the linked requirements in the modules of different integration levels;

4– the number of links in the modules of different levels of integration in accordance with client requests.

When other factors are formally defined, it is possible to define its breakdown into the five levels above mentioned. This will then permit the definition of a system capable of controlling specifications quality in LESD [Sla93].

The hierarchical model of software quality control allows flexibility in updating each one of these levels. A tool based on this model can be easily adapted to changes in defining factors, criteria, metrics, measures and their relationships.

CONCLUSIONS

The work done in the first phase of the LESD project, consisted of developing tools for the analysis of specifications written in natural language. To be more exact, this stage was devoted to the development of the syntactic and semantic parsers of these specifications; to the study of the knowledge necessary for their interpretation; to the design of an adequate knowledge representation system; and to the implementation of some reasoning mechanisms for evaluating factors involved in specifications quality. All this work was carried out in the aerospace domain.

Five quality factors of specifications have been considered (traceability, completeness, consistency, verifiability and modifiability). Traceability and evaluation techniques have been developed and studies on completeness are currently under way.

We plan to study other quality factors in the near future. A specifications quality control system based on software quantitative measurement will be created once a suitable evaluation algorithm of quality factors has been obtained.

We have also begun an environment study aimed at helping engineers write specifications and incorporating the quality control discussed in this paper at this early stage in a project's life.

REFERENCES

[AECMA89] Association Européene des Constructeurs de Matériel Aréonautique: "AECMA Simplified English, A Guide for the preparation of aircraft maintenance documentation in the international aerospace maintenance language", December 1989.

[BBC*92] Borillo M., Borillo A., Castell N., Latour D., Toussaint Y., Verdejo M.F.: "Applying Linguistic Engineering to Software Engineering: The traceability problem". In *Proceedings of the European Conference on Artificial Intelligence (ECAI92)*, pages 593-595, Vienna, Austria, August 1992.

[BGB*87] Briscoe T., Grover C., Boguraev B., Carroll J.: "The Alvey Natural language Tools Project Grammar: a Large Computational Grammar". ALVEY Documents, Cambridge Univ., Computer Laboratory, UK, 1987.

[BTB91] Borillo M., Toussaint Y., Borillo A.: "Motivations du project LESD". *Conference on Linguistic Engineering'91*, Versailles, France, January 1991.

[CC88] Cordes D.W., Carver D.L.: "Knowledge Base Applications within Software Engineering: A Tool for Requirements Specification". In *Proceedings of the First International Conference on Industrial & Engineering Applications of A.I. & E.S.*, Tullahoma, USA, pp. 266-280, June 1988.

[CC89] Cordes D.W., Carver D.L.: "Evaluation Method for User Requirements Documents". *Information and Software Technology*, vol.31, n.4, pp. 181-188, May 1989.

[CCG89] Carver D.L., Cordes D.W., Gautier N.: "Object-Based Measurement in the Requirements Specification Phase".

[Cor89] Cordes D.W.: "Improved Utilization of Requirements Document Information During System Specification", In Coulter N.S. and Ellis E. editors., *Proceedings of the 27th Annual Southeast Regional Conference*, pages 273-277, Atlanta, USA April 1989.

[DF84] Dorfman M., Flynn R.F.: "Arts – An Automated Requirements Traceability System". *The Journal of Systems and Software*, vol. 4, pp. 63-74, 1984.

[GKP*85] Gazdar G., Klein E., Pullum G., Sag I.: *Generalized Phrase Structure Grammar.* Harvard Univ. Press, Cambridge, UK 1985.

[Gro86] Grosz B.: "Attentions, Intentions, and the Structure of Discourse". *Computational Linguistics*, vol.12, n.3, July-September 1986.

[HOT91] Group H.O.T.: *HOOD Reference Manual*, HOOD, July 1991.

[IEEE84] "IEEE Guide to Software Requirements Specifications", ANSI/IEEE Std 729-1983, 1983.

[Pre87] Pressman R.S.: *Software Engineering: A Practitioner's Approach.* Mac Graw Hill, Nueva York, USA, 1987.

[RW91] Reubenstein H., Waters R.: "The Requirements Apprentice: Automated Assistance for Requirements Acquisition". *IEEE Transactions on Software Engineering*, vol. 17, n.3, March 1991.

[SA77] Schank R.C., Abelson R.P.: *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates Publishers, New Jersey, USA 1977.

[Sla92] Slavkova O.: *Métricas de software.* (In Spanish). Report LSI-92-6-T, Universitat Politècnica de Catalunya, Barcelona, Spain, 1992.

[Sla93] Slavkova O.: *Modelo para el control de calidad en LESD basado en la medición del software.* (In Spanish). Report LSI-93-26-R, Universitat Politècnica de Catalunya, Barcelona, Spain, 1993.

[SNM*88] Seki H., Nabika e., Matsumura T., Sugiyama Y., Fujii M., Torii K., Kasami T.: "A Processing System for Program Specifications in a Natural Language". In *Proceedings of the 21st. Annual Hawai International Conference on System Sciences.* Vol.II: Software Track, pages 754-763, ed. IEEE Comp. Soc. Press, Washington, USA, 1988.

[TB93] Toussaint Y., Borillo M.: "Natural Language in Software Engineering". In *Encyclopedy of Software Engineering*, John Wiley eds, 1993

[TBB*91] Toussaint Y., Borillo M., Borillo A., Castell N., Latour D.: "Applying Linguistic Engineering to Software Engineering". *26th Linguistic Colloquium*, Poznan, Poland, Setember 1991. (Published in *Linguistische Arbeiten*, n.293, pp. 209-217, Max Niemeyer Verlag, 1993)

[TC93] Tuells T., Castell N.: *The completeness problem in LESD.* Report LSI-93-51-R, Universitat Politècnica de Catalunya, Barcelona, Spain, 1993.

[Tou92] Toussaint Y.: *Méthodes Informatiques et Linguistiques pour l'aide a la Spécification de Logiciel.* Ph.D. Thesis. Université Paul Sabatier, Toulouse, France, 1992.

[WHH90] Wojcik R.H., Hoard J.E., Holzhauser K.: "The Boeing Simplified English Checker". *Conférence sur l'Intelligence Artificielle dans l'Aéronautique*, Toulouse, France 1990.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Recent Research Reports

LSI-93-47-R "Dealing with lexical mismatches", Carmen Soler and Ma. Antònia Martí.

LSI-93-48-R "Translation equivalence via lexicon: a study on tlinks", Anna Samiotou, Irene Castellón, Francesc Ribas, and German Rigau.

LSI-93-49-R "A class-based approach to learn appropriate selectional restrictions from a parsed corpus", Francesc Ribas.

LSI-93-50-R "Quality control of software specification written in natural language", Núria Castell, Olga Slavkova, Yannick Toussaint, and Antoni Tuells.

LSI-93-51-R "The completeness problem in LESD", Núria Castell and Antoni Tuells

LSI-94-1-R "Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.

LSI-94-2-R "Degrees and reducibilities of easy tally sets", Montserrat Hermo.

LSI-94-3-R "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI-94-4-R "Una modelizaciòn de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.

LSI-94-5-R "A multiple shooting vectorial algorithm for progressive radiosity", Blanca Garcia and Xavier Pueyo.

LSI-94-6-R "Construction of the Face Octree model", Núria Pla-Garcia.

LSI-94-7-R "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI-94-8-R "A transformation scheme for double recursion", José L. Balcázar.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es