UDC 004.4

I. BISCOGLIO, M. FUSANI, S. GNESI

ISTI – CNR, Pisa, Italy

CAN SAFETY BE OBTAINED THROUGH GOOD PRACTICES FOR REQUIREMENT WRITING?

Software Requirements analysis of quality characteristics as completeness, consistency and unambiguity assume an important role in the safety-critical software. We consider then the contributions to Natural Language (NL) Software Requirements Analysis as good practices or recommendations for producing high level quality NL requirements. From a survey of different approaches and solutions, it is possible to draw an incremental list of Good Practices (GP) for writing NL software requirements, making them understandable to their users, typically from the linguistic point of view, thus reducing the efforts for the analysis. The industrial experiences of two research/service laboratories about requirements analysis are compared with the suggestions of the literature and the technology, and the results are shown.

Keywords: *Natural Language, software requirements analysis, good practices*

Introduction

If we consider the different languages for expressing requirement specifications, Natural Language (NL) is a single modality of expression but, as it is well known, there are others. The requirement specifications can be written in *Structured Natural Language* (a restricted NL where the terminology is limited and templates can be used), or in *Semi-formal Language* (with graphical notations, precise syntax and a non-rigorous semantic) or in *Formal Language* (mathematics-based language with syntax and semantics formally defined). Among them, NL is the most common way to express software requirements: NL requirements can be easily communicated to the various stakeholders, technical and no-technical, before being used in the subsequent product development phases, and NL requirements facilitate communication among stakeholders.

However, NL requirements often are inherently ambiguous. Avoiding ambiguities can be considered an important step to produce quality safety-related software in time and with reasonable cost.

It is not uncommon that different users of the same requirements understand them in different ways. This situation may have detrimental effects on subsequent development phases: the cost of late removal of errors in requirements elicitation could be very high [5], as it implies much rework and maintenance.

Even when requirements have to be expressed with rigorous notation, which it is needed in most contracts and standards for safety-related systems, they are first conceived and written in Natural Language. The (mostly manual) re-writing process to have them as formal expressions is itself a critical job as it suffers from the risks of subjectivity and ambiguity.

Certainly, analyzing the quality of NL requirements is more difficult than analyzing semiformal or formal descriptions: the latters are less ambiguous than the former, but NL requirements are more pervasive and immediate in the communication process between requirements engineers and system stakeholders.

In general, requirement analysis is an expensive and time-consuming process. Even the few organizations that are aware of the risks due to poor requirements are reluctant to sustain the effort of analyzing their quality as it is considered a hardly sustainable resource investment.

As the goal of requirement analysis is establishing an agreed set of complete, consistent and unambiguous requirements, adopting good practices for writing NL requirements can be a relatively low cost against the benefits that may result. From this point of view, writing good NL requirements becomes an important investment for their subsequent analysis: if the requirements are well written their analysis will be faster and cheaper.

The problem of ensuring the quality of NL requirements remains a big deal. The process, as already mentioned, is heavily time consuming indeed, involving reviews and sometimes-partial prototyping.

Referring to recommendations about requirements standards [11, 12, 17], we can see that requirements specifications are expected to exhibit a list of quality *characteristics*, such completeness, consistency and unambiguity: incomplete, inconsistent, or even subject to misinterpretation, requirements may create problems both in software development and in operation with the finished product.

In this paper we try to take stock of the situation about NL software requirements analysis drawing an incremental list of Good Practices (GP) for writing high

quality requirements. In the following, we describe the experiences of the System and Software Evaluation Centre (SSEC) and the Formal Methods and Tools Laboratory (FMT) of the National Research Council at Pisa (CNR) about software requirements analysis in industrial contexts.

As we shall see, the activity of requirement analysis adopted by FMT Laboratory with the QuARS tool [10] highlights several potential sources of ambiguity and inconsistency in requirements that may case errors in the software development. By the experience of the SSEC, we also show that the resources dedicated to this activity by a set of examined developers are lower than those employed in other activities.

Section 1 introduces related works on NL requirements analysis. In Section 2, a list of GP for writing quality requirements is presented, and in Section 3 some industrial experiences are shown. Finally, conclusions close this work.

1. Related works

In order to produce quality software, starting from quality software requirements is a must, and, quality requirements are firstly good written requirements. As this need has been commonly recognised, several studies have defined methodologies and proposed tools for the analysis of NL requirements. In literature, two approaches emerge. The first approach is aimed at defining best practices or recommendations for both learning to write requirements less ambiguously and less imprecisely, and learning to detect ambiguity and imprecision in the written text (e.g. [2, 3, 11, 12, 15, 21]). The second approach, introducing automation in best practices adoption, is aimed at proposing tools that perform lexical, syntactical or semantic analysis, to detect ambiguity, inconsistency or incompleteness in NL requirements (e.g. [1, 9, 10, 14, 16, 18, 20, 22, 23]).

The first approach is rich, authoritative and clear, but it is likely to be theoretical and slightly abstract. The second approach is more detailed, but it is focused mainly on NL requirements analysis phase and not NL requirements writing phase. Besides the differences between the two approaches are related to type of detected ambiguity, scope, instruments, and solutions. Nevertheless, for both, the purpose is the same: avoiding or detecting ambiguity, inconsistency and incompleteness and, consequently, promoting understandability and clarity of users' needs.

2. A list of good practices (GP) for writing software requirements

From the above cited contributions and other sources that we mention soon, it is possible to draw an list of GP that can assist and help the NL software requirements writers. This GP list is incremental: new elements can come out from experience and NL requirements can be written always better. Besides, the GP presented here do not identify defined and closed categories of words or sentences, each category is not mutually exclusive of other categories. However, the use of GP could encourage the writing of more defined and less ambiguous NL requirements.

We derived our incremental list of GP from different sources of knowledge:

- the study of literature in the requirements analysis field;
- our research activity that led us to the design and the creation of the tool QuARS [10];
- the adoption of QuARS for analyzing requirements in projects [6] and requirements expressed as clauses of International Standards Requirements [4]. During our research activity we noticed how "good practices" in writing NL requirements can be derived from observing how frequent "bad practices" are reflected in software requirements documents. Thus, we have reversed the perspective: the analysis of problems in written text of NL requirements has returned a list of GP for writing quality requirements;
- our experience with many industries in the software product and process assessment / improvement [7, 8].

The experience of having approached the problem from the perspective of requirements analysis puts the scope of GP use in the initial phase of software lifecycle, particularly during the phase of writing of NL requirements.

A. The Good Practices

Adopting a domain glossary

Adopting a glossary is fundamental for writers of requirements as the meaning of words should be the same both to those *who write* and to those *who use*.

The idea of specialized glossaries [20] is adopted with the shape of having them organized into sections:

– acronym glossary, for every used acronym;

– agent glossary, for all valid agent entities (e.g. *actor*, *system*, *process*);

– action glossary, for all valid actions (e.g. *allow*, *generate*…);

– modal word glossary, for all valid modal words (e.g. *shall*, *should*…);

– remaining words glossary, for used words, mainly those related to the application domain the requirements are used.

Using terms *if and as* defined in the glossary

Do not use different terms to refer to the same thing.

Using correct grammar

Using the correct grammar the risk of ambiguity is more reduced.

Avoiding non-uniquely quantifiable words

Many words without precise quantifications, as *some, sufficient, minimum, maximum, immediately, periodically, etc.,* are ambiguous and not verifiable. These words could generate misunderstandings and errors, compromising the verification of compliance to requirements. A requirement should only contain measurable quantities.

Avoiding words holding inherent vagueness

The vagueness-revealing words, as *strong, acceptable, easy, adequate* and *difficult,* have no a unique interpretation, and it can change for different subjects.

Avoiding words used to express personal opinions or subjectivity-revealing words.

Examples of subjectivity-revealing wordings are *similar*, *simple, known, essential, usual.* In this context, it is important to avoid, for example, the adjectives.

Specifying always the subject of the sentence.

In a sentence, the presence of a generic subject expressed by demonstrative adjectives, pronouns, adjectives or preposition could make not clear the understanding of whole requirement.

Avoiding optional parts in a requirement.

In a requirement, we can find words or set of words, as *possibly, if needed, eventually, etc.* that introduce an optional part. The use of these components can generate the doubt of *what* to implement.

Avoiding a weak main verb

"Weak" modal verbs, as *can, could, may*, make the sentence not imperative, reducing the strength of the requirement.

Defining all references

If the sentence contains undefined or incorrect references, its understanding can be compromised.

Avoiding words identifying a class of objects without a specifyer of this class

It is possible that the sentence contains a word identifying a class of objects without a modifier (typically an adjective or a genitive) specifying an instance of this class, for example *access* (write access, remote access, authorized access…), *testing* (functional testing, unit testing…), *manual* (user's, maintainer's).

Theses cases of under-specification relate to a lack of completeness that is a characteristic of a good Software Requirements Specification [14].

Adopting the active form

The sentences in active form are direct, energetic, less ambiguous and make it clear who's doing what.

3. Industrial experiences on requirements analysis

Requirements Engineering is an area where much research work has been done for years. Problems analysis, solutions and proposals have appeared in many journals and conference proceedings. We have seen in the previous sections that literature surveys and known projects reports can give an insight on these speculative areas. But what has been happening in the practitioners' world?

In order to detect the real situation of many industries regard to requirement analysis, we have used the experience of our laboratories, the Formal Methods and Tools Laboratory (FMT) and the System and Software Evaluation Centre (SSEC), in interfacing with industrial software suppliers for providing services and for experimenting research results.

A. The FMT experience

The experience of the FMT regards the analysis of a large collection of NL requirements produced inside the EU/IP MODTRAIN project, in the MODCONTROL [19] subproject, from different project partners.

MODCONTROL addresses the standardization of an innovative Train Control and Monitoring System (TCMS) for the future interoperable European trains. Their requirements were collected in a SRD (System Requirements Document) with more than 5.700 requirements (exactly 5.777) categorized as:

– Functional Requirements (FREQ): Requirements for a TCMS function.

– System Requirements (SREQ): Requirements for devices carrying some functions (or sub-functions).

For the analysis, the 5.777 requirements (3.209 FREQ and 2.568 SREQ) were analysed by QuARS. The tool has produced rich reports of information about linguistic defects and writing style of NL requirements.

In both categories of analysed requirements there are *defective* (according to the QuARS' analysis) requirements (in FREQ, the defect rate[1] is 51% and in the SREQ the defect rate is 50%).

Each requirement can contain multiple errors. The types of detected errors (for their definitions, see [10])

[1] Defect Rate: number of requirements that present defects divided for total number of requirements [10]

recall the importance of the use of above cited GP in order to avoid them.

As we can see in the Table 1, in the analysed requirements there are sentences that have multiple subjects and then without a specified subject (multiplicity analysis), or sentences that contain non-uniquely quantifiable words and words holding inherent vagueness (vagueness analysis), or also sentences with weak modal verbs (weakness analysis), etc.

Table 1

FREQ and SREQ: Defects for Type

Defects	FREQ		SREQ	
	Defective Requirements	Errors	Defective Requirements	Errors
Optionality	35	47	23	29
Subjectivity	39	54	39	61
Vagueness	353	652	396	613
Weakness	128	164	54	61
Implicity	116	251	66	129
Multiplicity	847	2437	633	1809
Underspecification	129	190	68	120
Total	1647	3795	1279	2822

After having remove the "false positive",[2] the reminder requirements with these types of problems can create, in any case, misunderstandings and ambiguity among the stakeholders. The presence of the cited defects can have detrimental effects on the subsequent phases of the software process: as we said above, the cost of removing errors in requirements could be very high, requiring at least rework and maintenance.

B. The SSEC experience

The experience of SSEC regards the requirement analysis as object of independent software lifecycle process verification. After two decades of such work, we may conclude that only very few organizations perform requirements analysis as a specific and documented lifecycle activity.

From a repository of 26 ISO/IEC 15504 [13] assessments reports performed between 2001 and 2011, we did an explorative analysis of the assessment results about the Process Capability Level[3] "Performed" or "Level 1" of *System Requirement Analysis* process. This level is evaluated through indicators named *Base Practices (BP) or activities that, when consistently performed, contributes to achieving a specific process purpose* [13]. There are five BP:

- BP1 : Establish system requirements
- BP2: Analyse system requirements
- BP3: Evaluate and update system requirements
- BP4: Ensure consistency
- BP5: Communicate system requirements

Our aim is to observe the distributions of different BP and to detect relevant differences. As we can see in the figure 1, the most widely used modality (here modalities correspond to scores of practice performance) is that of *fully performed*: that is, the BP are practically fully achieved. Nevertheless, BP2, or *Analyze System Requirements*, and BP4, or *Ensure Consistency*, show some downturns and major variability. In a global positive situation of the level 1 of *System Requirement Analysis* process, these scores uncover weak points and scopes to be monitored: why these results? What do they mean? Is this a coincidence?

For both BP, the variability in the scores is not casual (we used One Sample Kolmogorov - Smirnov test obtaining the following results: 1,852 for BP2 and 2,025 for BP4 with $\alpha = 0.01$). Above all, it is very interesting that a statistically significant, strong and positive correlation between BP2, or *Analyze System Requirements*, and BP4, or *Ensure Consistency* was found [Spearman's rho Correlation Coefficient (SRCC) = 0.791, $P < 0.01$ level (2-tailed)]. Obviously, a strong correlation doesn't mean that a cause – effect relationship between BP2 and BP4 exist, but only that the organizations with the lower scores for BP2 are the same with lower scores also for BP4. Then, for BP2 and BP4 similar problems to be addressed and the situation deserves further insights.

It is fair to say that some actual requirements analysis is conducted for correctness and testability in the most mature suppliers organizations. Nevertheless, the quality aspects of consistency and non-ambiguity seem to be still far away from the industry targets.

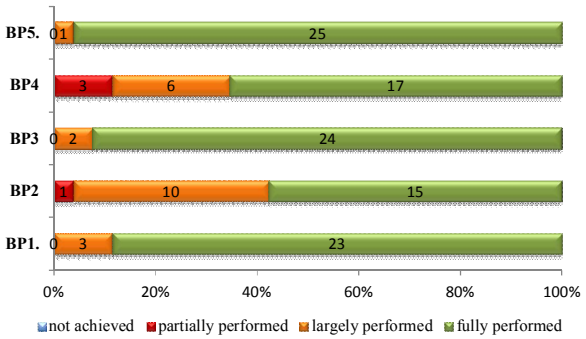

Fig. 1. Ratings for BP
in System Requirement Analysis Process

Conclusion and future work

In this paper, we provided an overview on NL software requirements analysis in literature and we have drawn an incremental list of GP for writing NL software requirements.

[2] False Positive: a requirement recognized as defective by the tool but considered acceptable by the user [10]

[3] Process Capability Level: a point on the six-point ordinal scale (of process capability) that represents the capability of the process; each level builds on the capability of the level below [13]

We have also presented the industrial experiences of two research/service laboratories about requirements analysis, together with the results of their analysis.

For the future, we consider the idea that the GP can be integrated in a project of textual editor that helps requirements engineers in writing software requirements. Particularly, for the future developments of this work, our projects are:

1. the implementation of the GP in a new part of our tool QuARS analyzer. The new part should be considered as the "editor" part of QuARS, able to assist the requirements engineers during the writing of NL requirements and to reduce the risk of their arbitrariness. The "editor" part of the tool does not exclude or reduce the importance of the "analysis" part: the editor should only provide "warnings" that flag potentially risky words or sentences, and suggest GP, but freedom and creativity in writing requirements would be guaranteed.
2. evaluating *if*, *how* and *how much* the above cited GP are implemented in the *System Requirement Analysis* Process in the process assessment activity of the System and Software Evaluation Centre (SSEC).

References

1. Ambriola, V. The Circe approach to the systematic analysis of NL requirements [Text]/ V. Ambriola. V. Gervasi // TR-03-05, University of Pisa, 2003.

2. Berry, D.M. From contract drafting to software specification: Linguistic sources of ambiguity [Text]/ D.M. Berry, E. Kamsties, M.M. Krieger // TR, University of Waterloo, 2003.

3. A new quality model for natural language requirements specifications [Text]/ D.M. Berry, A. Bucchiarone, S. Gnesi, G. Lami, G. Trentanni // In Proceedings of REFSQ, 2006.

4. An approach to Ambiguity Analysis in Safety-related Standards [Text]/ I. Biscoglio, A. Coco, M. Fusani, S. Gnesi, G. Trentanni //In Procedings of QUATIC 2010, Porto, Portugal, – P. 461-466.

5. Boehm, B.W. Software Engineering Economics [Text]/ B.W. Boehm // Prentice-Hall, Englewood Cliffs, NJ, 1981.

6. An experience in using a tool for evaluating a large set of Natural Language Requirements [Text]/ A. Bucchiarone, S. Gnesi, A. Fantechi, G. Trentanni, //Proceedings of ACM SAC, 2010. – P. 281-286.

7. Fabbrini, F. One decade of software process assessments in automotive: a retrospective analysis [Text]/ F. Fabbrini, M. Fusani, G. Lami// In: proceedings of ICCGI 2009 - Cannes, France, IEEE. – 2009. – P. 92 – 97.

8. Using software process assessment to manage the quality of suppliers: an experience in automotive [Text]/ F. Fabbrini, M. Fusani, G. Lami, E. Sivera //In Proceedings of the 15th ICSSEA 2002, Paris, France. – P. 29 – 35.

9. Fantechi, A. A Content Analysis Technique for Inconsistency Detection in Software Requirements Documents [Text]/ A. Fantechi, E. Spinicci //In: VIII Workshop on Requirements Engineering, 2005, Porto, Portugal.

10. Gnesi, S. An automatic tool for the analysis of natural language requirements [Text] / S. Gnesi, G. Lami, G. Trentanni //In International Journal of Computer Systems Science and Engineering. – 2005. – V. 20, N. 1. – P. 1-13.

11. Hooks, I. Writing Good Requirements [Text] / I. Hooks // Proceedings of the Fourth International Symposium of the NCOSE 2, San Jose, CA, 1994. – P. 197-203.

12. IEEE Recommended Practice for Software Requirements Specification [Text] // IEEE/ANSI Standard 830-1998, Institute of Electrical and Electronics Engineers, 1998.

13. ISO/IEC 15504. Information Technology Software Process Assessment [Text] // International Organisation for Standardization. Geneva Switzerland, 2006.

14. Automated review of natural language requirements documents: generating useful warnings with user-extensible glossaries driving a simple state machine [Text] / P. Jain, K. Verma, A. Kass, R.G. Vasquez // In Proceedings of ISEC 2009, ACM, 2009. – P. 37–46.

15. Kamsties, E. Detecting Ambiguities in Requirements Documents Using Inspections [Text] / E. Kamsties, D.M. Berry, B. Paech //In Proceedings of WISE, ed. M. Lawford and D. L. Parnas, Software Quality Research Lab at McMaster University in Canada, Paris, France, 2001. – P. 68–80.

16. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications [Text]/ N. Kiyavitskaya, N. Zeni, L. Mich, D.M. Berry //Requir. Eng. – 2008. – Vol. 13, No. 3. – P. 207–239.

17. Meyer, B. On Formalism In Specifications [Text]/ B. Meyer //IEEE Software. – 1985. – Vol. 2, No. 1. – P. 6-26.

18. Mich, L. Ambiguity measures in requirement engineering [Text] / L. Mich, R. Garigliano // In Feng Y., Notkin D., Gaudel, M., eds.: Proceedings of ICS2000, Sixteenth IFIP World Computer Congress, Beijing, Publishing House of Electronics Industry, 2000. – P. 39 – 48.

19. MODTRAIN: Innovative Modular Vehicle Concepts for an Integrated European Railway System [Electron resourse]. – Access an: http://www.modtrain.com. – 12.03.2012.

20. Verma, K. Requirements analysis tool: A tool for automatically analyzing software requirements documents [Text]/ K. Verma, A. Kass //In ISWC '08: Proceedings of ISWC 2008, Karlsruhe, Germany. – P. 751–763.

21. Wiegers, K. Software Requirements [Text] / K. Wiegers //Microsoft Press, 2003.

22. Wilson, W.M. Automated analysis of requirement specifications [Text]/ W.M. Wilson, L.H. Rosenberg, L.E. Hyatt // In Proceedings of ICSE '97, New York, NY, USA, ACM Press, 1997. – P. 161–171.

23. Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements [Text] / H. Yang, A. Willis, A.D. Roeck, B. Nuseibeh //Procedings of The 25th IEEE/ACM, ASE'2010.

Поступила в редакцию 12.03.2012

Рецензент: д-р техн. наук, проф., зав. каф. программной инженерии И.Б. Туркин, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков, Украина.

МОЖЕТ ЛИ БЕЗОПАСНОСТЬ БЫТЬ ОБЕСПЕЧЕНА С ПОМОЩЬЮ РУКОВОДСТВ ПО НАПИСАНИЮ ТРЕБОВАНИЙ?

И. Бисколью, М. Фузани, С. Нези

Анализ качественных характеристик полноты, согласованности и однозначности требований к программному обеспечению играет важную роль в обеспечении безопасности критического программного обеспечения. Мы считаем, что вклад в анализ требований к программному обеспечению на естественном языке является передовой практикой в производстве высококачественных NL-требований. Из обзора различных подходов и решений, возможно составить дополнительный список передовых практик для написания NL-требований к программному обеспечению, делая их понятнее для их пользователей, как правило, с лингвистической точки зрения, тем самым, снижая усилия, направленные на анализ. Промышленный опыт двух исследовательских/сервисных лабораторий в области анализа требований сравнивается с предположениями, описанными в литературе и показываются результаты сравнения.

Ключевые слова: Естественный Язык, анализ требований к программному обеспечению, руководства.

ЧИ МОЖЕ БЕЗПЕКА БУТИ ЗАБЕЗПЕЧЕНА ЗА ДОПОМОГОЮ ІНСТРУКЦІЙ З НАПИСАННЯ ВИМОГ?

І. Бісколью, М. Фузані, С. Незі

Аналіз якісних характеристик повноти, узгодженості та однозначності вимог до програмного забезпечення відіграє важливу роль у забезпеченні безпеки критичного програмного забезпечення. Ми вважаємо, що внесок в аналіз вимог до програмного забезпечення на природній мові є передовою практикою у виробництві високоякісних NL-вимог. З огляду різних підходів і рішень, можливо скласти доповнюваний список передових практик для написання NL-вимог до програмного забезпечення, роблячи їх зрозумілішими для їх користувачів, як правило, з лінгвістичної точки зору, тим самим, знижуючи зусилля, спрямовані на аналіз. Промисловий досвід двох дослідницьких/сервісних лабораторій в галузі аналізу вимог порівнюється з припущеннями, описаними в літературі, та відображається результат цього порівняння.

Ключові слова: Природна Мова, аналіз вимог до програмного забезпечення, інструкції.

Isabella Biscoglio is a post doc at CNR-ISTI since 2007. Her current research interests include study and development of web quality models and NL requirements analysis for safety-related Standards and Software requirements.

Mario Fusani has been with the National Research Council (CNR) of Italy since 1973. Since the 80's, he has been involved in the investigation of quality issues of software products and processes. In 1986 he founded the Software and Systems Evaluation Center, still operating at ISTI-CNR in Pisa. His current work is related with software certification and system safety.

Stefania Gnesi is director of research at CNR-ISTI since 2001, and she is head of the Formal Methods and Tool group. She has been chair of the ERCIM-FMICS working group from 2002 to 2005, and currently is deputy chair of the Formal Methods Europe (FME) association. Stefania Gnesi's current research interests include i) study and development of formal languages for the specification and verification of families of dependable systems; ii) application of model-checking techniques to complex case studies; iii) development of rigorous techniques for the analysis of Software requirements.