Semi-automatic Checklist Quality Assessment of Natural Language Requirements for Space Applications

Anderson Rossanez and Ariadne M. B. R. Carvalho
Institute of Computing
University of Campinas
Campinas-SP, Brasil
anderson.rossanez@gmail.com, ariadne@ic.unicamp.br

***Abstract*—Problems with the specification of software requirements documents are a common cause of software defects. In the space applications domain, such defects are very costly, especially when detected after deployment in the field. It is imperative to ensure that software requirements are well written to avoid the introduction of these defects. The quality of software requirements is frequently assessed via checklists, based on standards for space application software, and on problems found in previous projects. Given the importance of quality assessment, and the fact that it is manually performed by domain experts, we propose to develop a semi-automatic, natural language processing tool, to diminish the reviewer's effort in the assessment, and to reduce errors in this process.**

I. Introduction

Software defects, when detected at later stages of software development, can be very expensive. If they are detected when the software product is already deployed in the field, the cost is even higher and, depending on the application, the consequences of such defects can be catastrophic. A considerable number of software defects are related to problems in the software requirements, that could have been avoided if the Software Requirements Specification (SRS) document were well written and properly verified. In areas such as space applications, SRS documents are written almost entirely in Natural Language (NL), which makes them understood by a higher number of people, not only by the domain specialists. But, unfortunately, this may result in issues inherent to natural languages, such as ambiguity, incompleteness, misinterpretation, among others.

In order to detect problems in the software requirements, and to avoid finding software defects at the later development phases, researchers have proposed methods to assess the quality of the SRS documents. In the space applications domain, Véras et al. [1] provide an approach for SRS review, to be performed by domain specialists, guided by checklists consisting of several yes/no questions used to measure the quality of the software requirements. The strategy is composed of three types of checklists: the first one based on Packet Utilization Standards (PUS) [2], the second on Conformance and Fault Injection (CoFI) methodology [3], and the third on problems found in previous projects.

The PUS-based checklist is used to assure the compliance to the standard, defined by the European Cooperation for Space Standardization (ECSS). It contains questions elaborated from two services present in the standard: the telecommand verification service, and the on-board operations scheduling service.

The purpose of the CoFI-based checklist is to verify if the SRS handles situations related to software failures. In the CoFI methodology, Finite State Machines (FSMs) are generated from the SRS to produce test cases. Questions for the checklist were generated from the FSMs, which were, in turn, built from the two PUS services previously mentioned.

The objective of the error-based checklist is to detect typical specification mistakes. It comes from seven SRS problem reports of three real past projects of aerospace systems. Review Item Discrepancies (RIDs) were found and categorized: external conflict/inconsistency, lack of traceability, external incompleteness, incorrectness, internal conflict/consistency, application knowledge, readability, domain knowledge, and non-usage of standard. 22 questions were generated from each of the problem descriptions. The 22 questions from the error-based checklist are divided into four categories:

1) *Lack of Traceability*: 5 questions for detecting problems related to missing or wrong traceability, such as whether all requirements are traced to at least a system or an interface requirement, or to the correct system or interface requirement;
2) *Requirement Incompleteness*: 4 questions for detecting if the requirements do not contain terms like "TBD" (To Be Defined) or "TBC" (To Be Confirmed), and other indications of incompleteness;
3) *Incorrectness*: 10 questions aiming to detect words or expressions indicating incorrectness, such as "should", "might", or "in preference";
4) *Internal Conflict/Consistency*: 3 questions for checking if there are conflicts or other inconsistencies, like a reference to a figure or table which is not related to the requirement, or showing conflicting information with what the requirement states.

As stated in the work by Véras et al. [1], the checklists

978-1-5090-5120-5/16 $31.00 © 2016 IEEE
DOI 10.1109/LADC.2016.26

were manually used in three different projects, by six different domain specialists. They answered some questions differently due to different levels of expertise and, also, due to different interpretations of both the questions and the requirements. The discrepant answers direclty influenced the quality assessment of the SRS document under analysis. This problem was pointed out by the authors as an important issue for future work. We believe that, if this process were semi-automatic, the number of divergent answers from the specialists could be minimized, and the necessary effort in the process would be reduced.

II. Objectives

The aim of our work is to provide a semi-automatic method for helping in the quality assessment of SRS documents through Natural Language Processing (NLP). Specifically, we want to provide an automated solution for some of the problems described at Véras et al. [1], by developing a tool to be used in the quality analysis of natural language SRS documents. In this way, we expect to mitigate the influence of the human factor by reducing the effort in the process, preventing errors, and also, speeding up the quality assessment process. Our work will focus on the checklist generated from errors detected in past projects.

III. Literature Review

We searched the scientific literature for existing works regarding automated software requirements analysis using NLP. We found several works using NLP in requirements analysis, but none using NLP applied to the automation of checklists, similar to the one we propose here. We found works in three different areas: Quality Indicators, Quality Assessment Tools, and Model Generation.

A. Quality Indicators

These works describe some indicators used to measure the quality of NL SRS documents, such as ambiguity, incompleteness, lack of atomicity, among others. Some of them present methods for detecting such indicators using NLP techniques, such as the work by Lamar and Mocko [4], which deals with ambiguity, incompleteness, and lack of understanding; Ormandjieva et al. [5], dealing with ambiguity; Lash et al. [6], which deals with ambiguity, completeness, and vagueness; and Falessi et al. [7], dealing with the identification of equivalent requirements.

B. Quality Assessment Tools

These works describe tools that implement methods to assess the quality of natural language SRS documents, automatically or semi-automatically, using NLP techniques. Some of these tools are: NASA's ARM tool [8], which detects ambiguity, incompleteness, and lack of understanding; the NLARE tool [9], a modular tool to detect ambiguity, incompleteness, and lack of atomicity; the "Smells" detector [10], which identifies ambiguity and incompleteness; the work by Kiyavitskaya et al. [11], which presents a semi-automatic tool to help in the identification of ambiguity; the EA-Analyzer [12], which identifies conflicting requirements; and the Requirements Quality Analyzer (RQA) [13], which detects a wide range of problems on the requirements.

C. Model Generation

These works show a different use of NLP on SRS documents: the generation of models that are commonly used to produce test cases and source code. Models, such as UML diagrams or state machines, can also be very helpful for assisting the quality assessment of SRS documents. Some examples are: NAT2TESTSCR [14] and NAT2TESTIMR [15], which builds intermediate models for generating test cases; the work by Tichy and Koerner [16], building UML models for generating test cases and source code; Lee and Rine [17], generating models for helping to identify missing requirements; Wagner et al. [18], which identifies non-functional requirements from generated models; Popescu et al. [19], building models to identify ambiguity and inconsistency; Kof [20], using state machines for identifying incompleteness and inconsistencies; and Greghi et al. [21], generating extended finite state machines from SRS documents for generating test cases.

IV. Our Approach

We want to develop a tool to help the reviewer on applying the error-based checklist to assess the quality of SRS documents. This characteristic makes it different from the other tools that we found in the literature review, which perform software requirements analysis using NLP. Nevertheless, we will consider some characteristics and techniques used by those tools in this work. When applying the error-based checklist, the reviewer must answer 22 questions similar to the following:

1. *Are all the requirements without the word "should" indicating a non-mandatory feature of the system?*
2. *Are all the variables or numeric values with the proper units?*

Some of the questions will be automatically answered by the tool (e.g., question [1.]), in which the tool must check if the requirements contain words or expressions that should not be there. For the other questions (e.g., question [2.]), the tool will generate a warning and leave the question unanswered. The warnings will explain to the reviewer what to do for answering the question, either by providing processed data for helping the reviewer to answer it, or by describing the task the reviewer must manually perform.

V. Tool's Description

The tool will be divided into modules representing the question's categories from the error-based checklist. Each module will be responsible for specific actions, which will eventually result in either answers to questions, or warnings. The modules will use NLP tools available in the Stanford CoreNLP toolkit [22]. Figure 1 illustrates the tool's architecture in a high level.

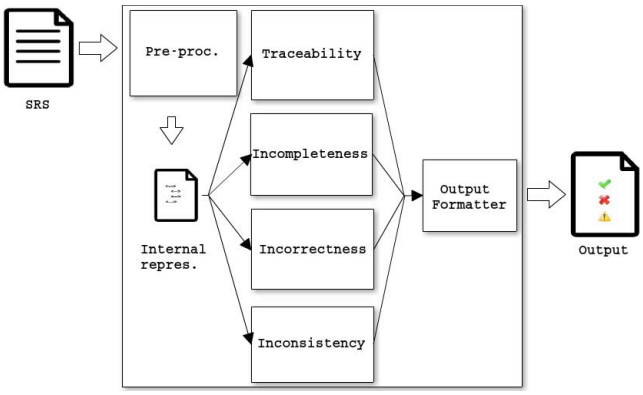

Fig. 1. Tool's architecture.

The first module is responsible for pre-processing the SRS document, which may be written in different formats, such as PDF, DOC, PPT, among others. Therefore, the first task is to convert the document into plain text for information extraction. This may be achieved through libraries such as Tika, from the Apache foundation, which extracts text and metadata from several document formats. Once plain text is obtained, the statements from the requirements can be retrived and converted to an internal standard representation, like XML (eXtensive Markup Language). Once the text is in the standard representation, it is used as input by the other modules.

The traceability module handles questions related to the lack of traceability from the error-based checklist. It must check if the requirements traceability matrix is complete, and raise warnings to help the reviewer in the traceability checking. To achieve this, the requirements traceability information must be retrieved from the internal representation, extracted in the pre-processing module.

In the incompleteness module, the questions from the requirement incompleteness category are considered. The module must raise warnings related to requirements with events and numeric values to be checked by the reviewer. Also, it must automatically answer questions related to terms that should not be present in the requirements, like "TBD" or "TBC".

The incorrectness module deals with questions related to the incorrectness category. In this module, most of the questions must be automatically answered, since they are mostly concerned with the identification of terms and expressions that should not be present in the requirements, indicating incorrectness. The Stanford TokensRegex, and a set of rules for identifying the undesired terms and expressions, will be used for this task.

The inconsistency module deals with questions from the internal conflict/inconsistency category of the checklist. It is responsible for raising warnings when finding references to images, tables, and lists in the requirements. Thus, the reviewer must check if such references are according to the requirements.

The last module is responsible for gathering all the warnings and answers from the previous modules and formatting the final output in form of tables, lists, or files.

VI. Conclusions

In the current state of our work, we can say that providing a semi-automatic tool to help reviewing SRS documents is feasible. Nevertheless, there is still a lot to be done in terms of the tool's development.

One point under investigation is the SRS document pre-processing, specifically the extraction of the requirements and other relevant information. Up to now, this task needs manual intervention, but we are investigating some alternatives, such as to use machine learning and named entity recognition combined with parse trees to automate the task. Besides, we will consider the methodology used in the work by Greghi et al. [21], whose input is also SRS documents written in different formats.

We also need to decide on the tool's output. It could be customized to fit different views, for example: a checklist view, showing the original checklist with the answers and warnings; a requirements view, showing the list of requirements where, for each requirement, we could expand a list showing all the answers and warnings from the checklist's questions; and views containing only requirements with warnings, positive or negative answers.

Finally, we must decide on the tool's validation. We expect to compare the results obtained from the manual analysis of the documents, performed by Véras et al. [1] using the error-based checklist, with the results obtained with the results of the analysis assisted by our tool. Also, we expect to run the tool on SRS documents containing injected defects, in order to see if they are properly caught either with warnings, or with automatically generated answers.

Acknowledgment

The authors would like to thank Paulo C. Véras, Emilia Villani, Ana M. Ambrósio, and Nuno Pedro Silva for their valuable inputs. This work is supported by the DEVASSES project (*DEsign, Verification and VAlidation of large scale, dynamic Service SystEmS* - http://www.devasses.eu).

References

[1] P. C. Véras, E. Villani, A. M. Ambrósio, R. P. Pontes, M. Vieira, and H. Madeira, "Benchmarking software requirements documentation for space application," in *Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security*, ser. SAFECOMP'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 112–125. [Online]. Available: http://dl.acm.org/citation.cfm?id=1886301.1886313

[2] E. Secretariat, "Ground systems and operations–telemetry and telecommand packet utilization ecss-e-70-41a," 2003.

[3] A. M. Ambrosio, E. Martins, N. L. Vijaykumar, and S. V. Carvalho, "A conformance testing process for space applications software services," *Journal of Aerospace Computing, Information, and Communication*, vol. 3, no. 4, pp. 146–158, 2006.

[4] C. Lamar and G. Mocko, "Linguistic analysis of natural language engineering requirement statements," in *Proceedings of the 8th International Symposium on Tools and Methods of Competitive Engineering, TMCE 2010*, vol. 1, 2010, pp. 97–111.

[5] O. Ormandjieva, I. Hussain, and L. Kosseim, "Toward a text classification system for the quality assessment of software requirements written in natural language," *SOQUA'07: Fourth International Workshop on Software Quality Assurance - In conjunction with the 6th ESEC/FSE Joint Meeting*, pp. 39–45, 2007. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-41149126094&partnerID=40&md5=7b3773dc0cbf71825f30bef11745c9af

[6] A. Lash, K. Murray, and G. Mocko, "Natural language processing applications in requirements engineering," *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2, no. PARTS A AND B, pp. 541–549, 2012. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84884646680&partnerID=40&md5=8b495498317bb41fb971c6ceb6e44d34

[7] D. Falessi, G. Cantone, and G. Canfora, "A comprehensive characterization of nlp techniques for identifying equivalent requirements," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 18:1–18:10. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852810

[8] N. Carlson and P. Laplante, "The nasa automated requirements measurement tool: A reconstruction," *Innovations in Systems and Software Engineering*, vol. 10, no. 2, pp. 77–91, 2014. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84900478117&partnerID=40&md5=ab43a51dd0226b01691efd1dc373db81

[9] C. Huertas and R. Juárez-Ramírez, "Nlare, a natural language processing tool for automatic requirements evaluation," in *Proceedings of the CUBE International Information Technology Conference*, ser. CUBE '12. New York, NY, USA: ACM, 2012, pp. 371–378. [Online]. Available: http://doi.acm.org/10.1145/2381716.2381786

[10] H. Femmer, D. M. Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer, "Rapid requirements checks with requirements smells: Two case studies," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE 2014. New York, NY, USA: ACM, 2014, pp. 10–19. [Online]. Available: http://doi.acm.org/10.1145/2593812.2593817

[11] N. Kiyavitskaya, N. Zeni, L. Mich, and D. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, pp. 207–239, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00766-008-0063-7

[12] A. Sardinha, R. Chitchyan, N. Weston, P. Greenwood, and A. Rashid, "Ea-analyzer: automating conflict detection in a large set of textual aspect-oriented requirements," *Automated Software Engineering*, vol. 20, no. 1, pp. 111–135, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10515-012-0106-7

[13] G. Genova, J. Fuentes, J. Llorens, O. Hurtado, and V. Moreno, "A framework to measure and improve the quality of textual requirements," *Requirements Engineering*, vol. 18, no. 1, pp. 25–41, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00766-011-0134-z

[14] G. Carvalho, D. Falcão, F. Barros, A. Sampaio, A. Mota, L. Motta, and M. Blackburn, "Nat2testscr: Test case generation from natural language requirements based on scr specifications," *Science of Computer Programming*, vol. 95, no. P3, pp. 275–297, 2014. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84908357432&partnerID=40&md5=edbecad3633d1b9d7190c77de555357f

[15] G. Carvalho, F. Barros, F. Lapschies, U. Schulze, and J. Peleska, "Model-based testing from controlled natural language requirements," *Communications in Computer and Information Science*, vol. 419 CCIS, pp. 19–35, 2014. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84904625090&partnerID=40&md5=910a7a3376ba60021ecd161f4df6a2e6

[16] W. Tichy and S. Koerner, "Text to software: Developing tools to close the gaps in software engineering," *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010*, pp. 379–383, 2010. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-79951634595&partnerID=40&md5=c3abe9b0ac8ea04c6857ab18aa9228ab

[17] S. W. Lee and D. C. Rine, "Missing requirements and relationship discovery through proxy viewpoints model," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004, pp. 1513–1518. [Online]. Available: http://doi.acm.org/10.1145/967900.968203

[18] S. Wagner, F. Deissenboeck, and S. Winter, "Managing quality requirements using activity-based quality models," in *Proceedings of the 6th International Workshop on Software Quality*, ser. WoSQ '08. New York, NY, USA: ACM, 2008, pp. 29–34. [Online]. Available: http://doi.acm.org/10.1145/1370099.1370107

[19] D. Popescu, S. Rugaber, N. Medvidovic, and D. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," in *Innovations for Requirement Analysis. From Stakeholders Needs to Formal Designs*, ser. Lecture Notes in Computer Science, B. Paech and C. Martell, Eds. Springer Berlin Heidelberg, 2008, vol. 5320, pp. 103–124. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89778-1_10

[20] L. Kof, "Translation of textual specifications to automata by means of discourse context modeling," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, M. Glinz and P. Heymans, Eds. Springer Berlin Heidelberg, 2009, vol. 5512, pp. 197–211. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02050-6_17

[21] J. G. Greghi, E. Martins, and A. M. B. R. Carvalho, "Semi-automatic generation of extended finite state machines from natural language standard documents," in *Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on*, June 2015, pp. 45–50.

[22] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010