Towards Computer-Aided Software Requirements Process

Marina Reich
Technische Universität Chemnitz
Airbus Defence and Space GmbH
Manching, Germany
marina.reich@airbus.com

Tatiana Chuprina
fortiss GmbH
Mnchen, Germany
chuprina@fortiss.org

Vivek Nigam
fortiss GmbH
Mnchen, Germany
nigam@fortiss.org

***Abstract*—It is a consensus that projects that start from good requirements have greater chances of success. Clearly specified requirements lead to better feature development, validation and verification. Unfortunately, many avionics product defects can be traced back to flaws in the Software Requirement Process (SRP), such as errors in requirements and in the communication between requirement engineers (SRE) and reviewers (SRR). This short-paper reports on our first steps towards a Computer-Aided SRP (CSRP). CSRP relies on three key ideas, leading to better quality requirements: Domain Specific Requirements (DSRs), Automated Requirement Quality Checks, and Structured Feedback. We illustrate CSRP with a use case on the development of requirements for a software in an embedded avionics system.**

I. Introduction

The avionics industry is making efforts to meet the increasing needs of handling requirements for complex avionics software [1]. It has been long known that many system and software defects can be traced back to badly written requirements [2]. This is also the case with avionics software [3], [4]. Indeed, the cause of many avionics software defects have been traced back to requirement engineering (RE) errors, such as, errors in recognizing requirements, requirements not well documented, and communication errors within and between development teams [4].

Therefore, a proper RE process that reduces the chances of flaws is important for the avionics industry. For the process design we consider two important observations in industry (gained within the ASSET[1] project): First, the role of the Reviewer (SRR) needs stronger guidance. The Reviewer (SRR) can be a company- internal department ensuring that his requirements are correctly specified or further developed by an another department or an external company within the next development stage. We consider the special transition from System Requirements to High Level Software Requirements. Today, the SRR is involved in the development of the requirements of the next development stage and supports with clarification of provided requirements, but do not follow a structured procedure. Second, the particular software in the domain of avionics needs to comply with the objectives of the DO-178C/ED-12C [5] in order to obtain certification. This guidance presents the activities to be performed during SRP with which the objectives shall be fulfilled. Today, the activities for SRP are not broken down into precise tasks for job roles (SRE, SRR) in the industrial context. Also, the activities are not interlinked precisely with the objectives and the quality of requirements.

Unfortunately, as described above, requirements many times do not meet the demanded quality. We believe that there are many reasons for this among which are the following:

- Tight development deadlines leading to sloppy RE;
- Requirements are normally written in natural language. Although the language used satisfies some criteria, such as avoiding the use of words, like “should”, the nature of natural language leads many times to ambiguities and to requirements with missing information;
- Reviewer feedback, such as request for corrections, is not standardized by existing process setup. Also, the feedback is written in natural language, which may lead to feedback misunderstandings.

We claim that the quality of requirements and of RE can be improved with a sophisticated and computer-aided process. This paper describes our initial steps towards the mitigation of the problems described above by proposing a Software Requirements Process (SPR), called Computer-Aided SRP (CSRP), that is supported by automated mechanisms for requirements quality assurance and structured feedback. Our CSRP relies on the following three key ideas:

- **Domain Specific Requirements** (DSR) which are requirements containing formal or semi-formal data. Different types of DSRs are associated to different data. The data is written by SREs using forms and graphical interfaces which *resemble the usual structure of the requirement*. Instead of using natural language, the data in a DSR have precise meaning, enabling a number of automated quality checks;
- **Automated Quality Checks**, which rely on the (semi-) formal data in DSRs to check automatically for errors in the quality of requirements. This not only facilitates the reviewing work, but also turns the process more agile, as SREs can obtain feedback on the quality of requirements before interacting with the SRRs. We illustrate the checks here with two types of DSRs (Mode and Signals);
- **Structured Reviewer Feedback** from the SRRs to the

[1]The ASSET Project (Avionic System Software Embedded Technology) was financially supported by the Aeronautical Research Program V (LuFo V) of the German Federal Ministry of Economic Affairs and Energy.

SREs. Instead of simply communicating the review of requirements in natural language only, reviews in CSRP are categorized according to the quality parameter that is not being satisfied. We argue that such feedback reduces the communication errors within the requirement engineering processes, thus leading to better quality requirements.

We illustrate the CSRP by a simple proof of concept example of an embedded system examined in the ASSET project. We start by reviewing relevant literature in Section II. We describe the general process of the CSRP in Section III and discuss DSRs in Section IV. We describe a case-study of an embedded component for avionics in Section V. We conclude this short-paper by pointing out to future work in Section VI.

Finally, we point out that we implemented part of the machinery described in this paper in the tool AutoFOCUS3 [6].

II. Related Work

We can roughly classify Requirement Engineering (RE) literature into three different categories: Natural-Language Based Requirements (NLBR); Formalized Requirements; and Pattern-Based Requirements. We detail this literature below.

The literature in NLBR uses textual requirements being closer to the actual industry practice. Therefore, the proposed methods implementing a type of Controlled Natural Language based on Natural Language Processing (NLP), e.g., ensuring active voice, can be directly applied in existing practices. Achour classifies in [7] methods in NLP in three categories: lexical, syntactic and semantic. The general goal is to interpret the meaning of sentences by identifying sentence objects and constructing relations among these objects. They can check, for example, when some sentences are ambiguous or do not comply with RE practices. Some commercial tools are offered to industry already, e.g., [8], [9]. However, given the informal nature of natural language, it is not possible to capture the exact intended semantics of textual requirements.

The literature on formalized requirements assumes that requirements are specified in some formal language, such as Linear Temporal Logic [10] formulas. The advantage of this approach is that a large number of properties can be formally verified using techniques, such as model-checking. The disadvantage is these methods are two-fold: Firstly, writing such requirements is very cumbersome and therefore, have not been widely adopted in practice. Secondly, the types of requirements that can be written is also constrained by the formal language used. For example, non-functional requirements cannot be specified in LTL.

The third type of approach, in which this paper is also included, is the Pattern-Based Requirements. Instead of writing requirements as formal specifications, SREs use pre-established requirement patterns. Patterns may be textual-like specifications using a Controlled Natural Language [11]–[13], e.g., the sentence written with a particular structure; Patterns may also be graphical representations, using diagram notations, such as UML and Sys-ML, and inspired by model-based approaches [14], [15].

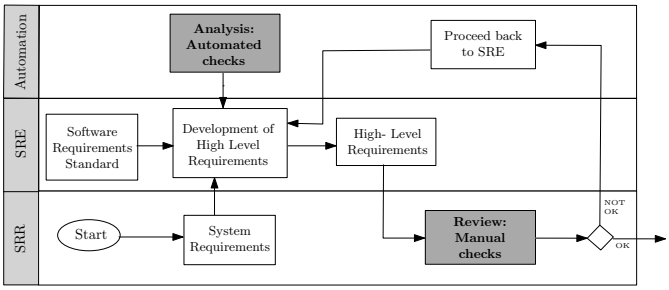

Fig. 1. Computer-Aided Software Requirements Process for HLRs. The Process for LLRs is similar and follows once HLRs have been developed.

Textual-like patterns [11]–[13] provide a general set of textual patterns that can be mapped to formal specifications expressed in LTL and its extensions. Requirements are written in natural language, but still become formalized. The disadvantage is that the types of patterns is constrained by the underlying formal language.

This paper is inspired by the model-based Integrated Requirements Analysis (MIRA) framework [14], [15]. A key difference is that, while MIRA proposes methods for formalizing textual requirements, by using Message Sequence Charts (MSC) or even logics, such as CTL, here we identify templates that *resemble closely usual textual requirements*. Thus, by using our templates, a SRE does not have significant additional effort, such as writing MSCs or CTL formulas, in developing requirements, while still profiting from automated quality checks. Finally, our methodology is more general than MIRA, as it can be used in development processes different from model-based Engineering processes.

In comparison to NLP methods we do not analyze for syntactical quality, but we structure requirements to ensure their quality using automated methods based on well defined semantics. The requirement's structure and its associated semantics is domain specific. This allows us for a finer analysis of requirements, e.g., consistency and completeness.

III. Computer-Aided Software Requirements Process

As pointed out in the Introduction, key sources of problems involving requirements are errors in communication and errors in requirements. We strongly believe that many of these errors can be mitigated by adding automated support to the SRP.

Figure 1 depicts the workflow of our CSRP. It complies with the DO-178C/ED-12C objectives. In particular, System Requirements and Software Requirements Standards are given as input to the SRP. High-Level Requirements (HLRs) and Low Level Requirements (LLRs) are the output of the process. Moreover, the HLRs and LLRs shall satisfy some quality criteria, *e.g.*, unambiguity, verifiability.

SREs and SRRs carry out the process, where SREs develop requirements, while SRRs review requirements in order to assure their quality. These roles are reflected in Figure 1. Our process contains a third role of automation. Automation includes simple checks, such as (those proposed in [15]) checks whether each requirement has an ID, an author, a

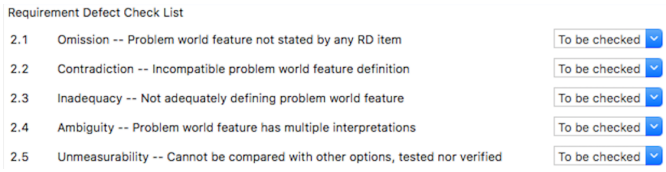

Fig. 2. Illustration of a Structured Reviewer Feedback template.

rationale, but also more complex ones, like automated checks for ambiguity, consistency, completeness. We illustrate some of these automated checks in Section IV.

From the given system requirements, SREs write HLRs. These HLRs can be textual requirements or DSRs which resemble the textual requirements, but enable more automated checks. HLRs are not manually reviewed by SRRs until all suitable automated checks have been satisfied. Whenever an automated check is not satisfied, an error message is returned proving further details for the error. In this way, the SREs can immediately have a feedback on the quality of the written HLRs without the need of the SRRs assessment. This saves time in the SRP as it reduces the number of reviewing cycles in the SRP and also ensures the quality of requirements.

Once all checks have been satisfied, the SREs can forward the developed HLRs to the SRRs for reviewing. The SRRs carries out reviewing that is not captured by the automated checks. For example, the consistency of names or the language used in the textual requirements. In order to guide the reviewer and improve the process, we also developed predefined feedback templates. A partial template is depicted in Figure 2. Instead of writing reviews as a document, SRRs fill the feedback template containing more structure. This helps to ensure, that SRRs have taken into account all relevant quality parameters and also helps SREs better understand the feedback provided, thus mitigating errors in communication.

LLRs are developed once the HLRs are approved, following same strategy, namely, SREs develop requirements with automated support and reviewer feedback. However, the types of automated checks may be different as LLRs shall satisfy other quality parameters, e.g., contain enough information for development and verification.

IV. Domain Specific Requirements and Automated Checks

The greater the number of automated checks, the greater is the chance of obtaining higher quality requirements. However, the number of automated checks depends on the level of formalization of the requirements. On the one hand, informal, textual requirements only allow for simple checks, such as the presence of rational, id numbers, while more complicated checks such as consistency, completeness has to be carried out manually. On the other hand, formal requirements written in a formal languages, such as LTL specifications [10], allow for more complicated checks to be carried out automatically using automated tools. However, formal requirements are not widely used in practice, as they are far more laborious to write, requiring many times expertise knowledge.

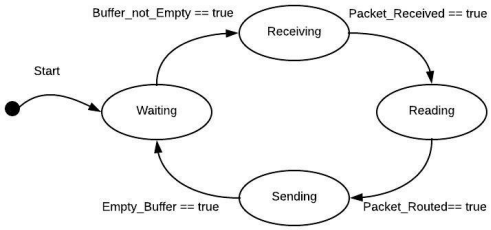

Fig. 3. DLUF Mode Automaton

Our hypothesis here is that specially tailored requirement templates for specific domains, called *Domain Specific Requirements* (DSRs), can support a large number of automated quality assurance checks and at the same time be used in practice. This is because, while data is structured or even formalized, the template used in DSRs resembles closely the textual requirements written for the particular domain.

Different DSRs will have different templates. *The challenge is, therefore, to develop requirement templates that resemble textual requirements, but at the same time allow for greater number of automated checks.* We illustrate DSRs next with two examples, including their supported automated checks.

A. Two Examples of DSRs

Our first example of DSR is for the specification of the modes of an embedded system and the second example for the specification of the signals of an embedded system.

a) Mode DSR: Mode requirements normally contain a diagram resembling a state transition system. Modes are represented in the diagram as states and arrows from states are drawn representing the allowed mode transitions. Moreover, the conditions when the transitions can occur are specified (normally, in LLRs, though) in text. In embedded systems, conditions are based on the values of the signals of the embedded system. Figure 3 depicts such a diagram.

Unfortunately, the diagram with the state transition system does not allow any automated checks to be carried out as the diagram cannot be parsed by a machine. We propose instead to directly write the diagram as a state-automaton using the machinery provided by AutoFOCUS3. *The effort required by an SRE to write the state-automaton is similar to the effort of drawing the diagram*. However, the state-automaton can be parsed by AutoFOCUS3 and automated checks can be carried out, which were not possible with the diagram alone.

b) Signals DSR: Signals are normally specified textually by sentences of the following form: *"The system shall contain an input signal called s of type int and ranging from* $[0, 255]$*."* This textual requirement contains the name of the signal, the type of data transmitted in the signal and its range. A signal requirement that does not contain, for example, the type of the signal, would be ambiguous, as it would allow for both integers and floats to be transmitted. Unfortunately, it is not possible to perform this check automatically as a machine cannot parse, in general, this information from the textual requirement.

The signals DSR contains a table of the form:

Name	I/O	Type	Lower Range	Higher Range
s	*input*	*int*	0	255

Notice that the effort required by the SRE to fill in this table is similar to the effort required to write the sentence above.

B. Automated Checks

Some checks can be performed taking into account each DSR individually, while others take into account both DSRs. In both cases, some quality criteria, such as Comprehensibility, Modifiability, Good Structuring are inherited by the formal nature of state automaton in Mode DSRs and of signals table in Signal DSRs.

We describe some automated checks for Mode DSR and elide the checks for Signal DSRs due to space limitations. The checks use standard machinery from automata theory [16].

- **Ambiguity**: It shall not be possible to transit from a mode to more than one mode with the same conditions. This property is reduced to checking that the mode state automaton is deterministic [16].
- **Verifiability**: It is possible to verify each transition of the state automaton without to modify it. In particular, for each transition, it is possible to state the starting mode, the values of the signals and the expected final mode.
- **Consistency**: This quality criteria can also be checked by using machinery available for state-automaton. For example, we check whether all modes can be reached using state reachability algorithms available in automata theory [16].

For Signal DSRs, we can also check for these quality requirements using the information available in the signals table. For example, it is not consistent if the lower bound is greater than the upper bound. Or it is ambiguous if some fields in the table are missing.

Finally, for automated checking of the **Completeness of the Mode DSR** we need to use the data available in the Signals DSR. For example, if a signal s can have values 0,1,2 and a mode has transitions considering only $s == 1$ and $s == 0$, but no transition with $s == 2$, then it is not complete.

V. Case Study DLUF

During the ASSET project, one team was engaged with the development of the Data Link Upload Feed (DLUF) and took the role of SRE. The customer of the developed software was the SRR.

DLUF description: In the context of avionic systems, the Data Link may be used by several so-called Data Link user to transmit data packets. One or several Data Links users, or more exactly the data they send, may have a higher priority than others. Due to the data rate budget, determined by properties of the Data Link in its selected operation mode (which may also dynamically change), packets from lower priority could prevent packets of higher priority to be transmitted if the budget is exceeded.

We summarize the results of the two main aspects of our investigations, the CSRP and the DSR usage:

SRP: The CSRP significantly reduced the personal meetings. The review took 36% of the total development time. 8% of the time were spend with the improvements according to SRRs feedback.

DSR: To all requirements at least one DSR was assigned. Several System Requirements contained information that needed to be split into two or more different DSRs.

VI. Conclusions and Future Work

This paper discusses our initial results towards the development of a CSRP. In particular, this process is supported by Domain Specific Requirements, Automated Checks and Structured Reviewer Feedback. One key motivation which differentiates us to other proposals is the use of requirement templates that closely resemble textual requirements written in practice, but allow for automated checks. We describe our first experiences with this approach in an avionics case-study.

As future work, we are identifying a suit of DSRs expanding on the ones we described here. In particular, we are aiming for DSRs used for non-functional requirements, such as performance requirements. We are also currently further developing our AutoFOCUS3 implementations to support new DSRs and new automated checks. Finally, we are also envisioning a refinement of the process described here with information on the dependency of DSRs. For example, the checking of the completeness of the mode DSR is only possible once the signals DSR is available. Including these dependencies into the process will also help guide SREs and SRRs in order to profit the most from automated checks available.

References

[1] F. A. A. U.S. Department of Transportation, "Requirements engineering management handbook, final report," 2009.

[2] S. P. Miller, A. C. Tribble, M. W. Whalen, and M. P. Heimdahl, "Proving the shalls," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, no. 4-5, pp. 303–319, Aug. 2006.

[3] T. Nakajo and H. Kume, "A case history analysis of software error cause-effect relationships," *IEEE Trans. on Soft. Eng.*, 1991.

[4] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," 2001.

[5] "RTCA DO-178C / EUROCAE ED-12C: Software Considerations in Airborne Systems and Equipment Certification," Standard, 2011.

[6] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz, "AutoFOCUS 3: Tooling concepts for seamless, model-based development of embedded systems," in *ACES-MB '15*, 2015.

[7] C. B. Achour, "Linguistic instruments for the integration of scenarios in requirement engineering 1 (position paper)," 1997.

[8] Ravenflow, "Ravenflow," 2018. http://www.ravenflow.com/

[9] the REUSE company, "Requirements quality suite," website, called 2018. hhttps://www.reusecompany.com/requirements-quality-suite

[10] A. Pnueli, "The temporal logic of programs," in *FCS*, 1977.

[11] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in ICSE '99. 1999.

[12] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar," *IEEE Transactions on Software Engineering*, vol. 41, no. 7, pp. 620–638, July 2015.

[13] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (EARS)," in *RE* 2009.

[14] A. Vogelsang, "Model-based requirements engineering for multifunctional systems," Ph.D. dissertation, TUM, 2015.

[15] S. M. Teufl, "Seamless model-based requirements engineering: Models, guidelines, tools," Ph.D. dissertation, TUM, 2017.

[16] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. 2006.