Detection of Defective Requirements using Rule-based Scripts

Hussein Hasso
hussein.hasso@fkie.fraunhofer.de
Michael Dembach
michael.dembach@fkie.fraunhofer.de
Hanna Geppert
hanna.geppert@fkie.fraunhofer.de
Daniel Toews
daniel.toews@fkie.fraunhofer.de

Fraunhofer FKIE
Fraunhoferstrasse 20
53343 Wachtberg

Abstract

In this paper we present our experience with rule-based detection of defects in requirements expressed in German language. It will elaborate on the types of defects, how they can be described using linguistic formalisms, and why the rule-based approach may be promising. Furthermore, we provide insights on two specific rules and show some results of a first evaluation.

1 Introduction

It is common practice in Requirements Engineering to elicit requirements from stakeholders in natural language. However, writing such requirements can be a difficult task, as certain factors, such as insufficient language skills or a lack of domain specific knowledge, may distort or omit information. This effect is referred to as defect in the literature [R+14] if it harms the requirements quality. Since a badly maintained requirement set can lead to confusion and thus more work, the Quality Assurance (QA) of requirements aims to detect these defects.

Many rules can be found on how to write good requirements. The German SOPHIST GROUP has published a set of regulations that contains 18 linguistic rules for the systematic analysis of requirements [R+14]. If one of these 18 rules is broken, the requirement in question possibly has a defect. Checking a set of requirements against these eighteen rules quickly becomes cumbersome, which undermines the benefit of having collected them. The question therefore arises, whether requirements can be scrutinized automatically. Our work concentrated on how to do this: We reviewed the 18 rules of the SOPHIST GROUP and additional 4 rules specified in [Hue03]. We then tried to formulate certain linguistic patterns, which check if one of the rules has been violated, using a rule-based scripting language. Those patterns will be called NLP patterns as they are called in the literature [FGR+18].

The decision to use rule-based scripts was made because they can be quickly produced and easily maintained with the use of certain linguistic knowledge. German phenomena in particular are conveniently formalized due to the strict word order of the language, a point that will be elaborated further in chapter 3.2.

Our work is motivated by [GFL+13] and [SBA13]. In [GFL+13], the authors synthesized different lists of desirable properties for requirements and arranged them graphically according to their mutual influence. Their arrangement indicates that high precision and atomicity of requirements will have a positive effect on multiple other desirable properties. Same characteristics are depicted in [SBA13]. It thus seemed beneficial to check and

improve the precision (all terms used are concrete and well defined [GFL+13]) and atomicity of requirements at first for the quality assurance task.

2 Related Work

Fabbrini et al. [FFGL01] introduced QuARS (Quality Analizer of Requirment Specifications), a tool for the analysis of requirements. Its aim is to automatically detect what they refer to as linguistic inaccuracies. They have defined a quality model against which requirements could be checked, to remove linguistic inaccuracies as much as possible. The Quality model is composed of high-level quality properties that are linked with keywords and syntactic elements that serve as indicators to find potential problems in the requirements. The results showed that around 50% of the total number of requirement sentences were marked as having a potential defect and up to 55% of the particular defect type "multiple" could be detected.

Ferrari et al. [FGR+18] showed that NLP technologies can be used to develop "in-house-tools" for defect identification in the industry. They compared two different approaches. Traditional manual defect detection used in industry for requirements analysis was compared with an analysis performed with NLP Patterns. Additional experiments performed with SREE [TB13] showed the detection of further defects, which had not been detected in the first iteration. Compared to our work, the authors used a different set of NLP patterns. Additionally, our NLP patterns are designed for German requirements and thus consider a completely different structure.

3 Past Research on Quality Assurance for Requirements

3.1 Linguistic Rules and Defects

Not all 22 rules of the SOPHIST GROUP and [Hue03] were considered to be suitable. Some rules required a semantic analysis as well as analysis of the whole requirement stock. These were consciously excluded, as we wanted to analyze how well a purely rule-based approach would peform. Some rules seemed to implicate each other. E.g. rule 3 [R+14] asks the Requirement Engineer to resolve nominalizations that are not clearly defined, while rule 12 [R+14] warns against the use of vague nouns; as nominalizations are nouns, rule 12 implies rule 3.

We identified ten rules as being relevant for our work. In the following, the term "defect" will be used in this paper as a synonym for the violation of one or more of these ten rules.

1. No use of passive voice: The passive voice is to be avoided as it may not specify, who is supposed to perform the action described: "The data **has to be entered** every morning." The sentence would be more informative in active voice: "The administrator **has to enter** the data every morning."

2. No empty verb phrases: Empty verbs are verbs of such very broad meaning that they transfer the expression of the actual process to a noun. They should not be used, as the process in question should be derivable from the main verb of a requirement. E.g.: "The system should **perform a data transfer** regularly." In this example, the empty verb would be "perform". Better: "The system **should transfer** data regularly."

3. No incomplete conditions: Requirements with incomplete conditions describe the desired behavior for a specific case, but they do not explain the desired behavior for the default case. E.g.: "**In a state of emergency**, the system needs to transfer data via radio." Better: "**In state of emergency**, the system needs to transfer data via radio, **in all other cases** transfer via cable is sufficient."

4. No redundant subordinate clauses: Redundant subordinate clauses explain aspects that are irrelevant for the requirement. E.g.: "The administrator needs to change data at any time **in order to help the user with his problems**." It would be better to delete the subordinate clause completely and, if necessary, store this information in an additional note.

5. Use conditional clauses instead of temporal clauses: Temporal clauses can be confusing in the context of requirements, because their function may not be clear. In most cases, they are actually to be understood as a condition, in which case a conditional clause should be used. E.g.: "**While** the system is booting up, data mustn't be sent." Better: "**If** the system is booting, data mustn't be sent."

6. No incomplete comparisons: A comparison is incomplete if there is no value for reference. E.g.: “The system needs to be faster”. Better: “The system needs to be faster **than 1000 MB/s**.”

7. Be careful with universal quantifiers: Such quantifiers might indicate a defect and should at the very least be questioned. E.g.: “**All** users should have access to the database. ” Should all of the users really have access?

8. No indefinite article: This rule applies specifically to requirements written in German since the indefinite article and the numeral ‘ein’ and ‘one’ are homonymous in German. This can lead to confusion and can, in most cases, be avoided by using the definite article, since words like “user ” refer to a certain role and not to one unspecified person, as the indefinite article suggests.

9. Be atomic: A requirement is not atomic if it consists of two or more requirements. The use of “and” in certain positions is one indicator, among others. E.g.: “The application should transmit data via radio and run on every operating system.” It would be better to write two separate requirements in this case.

10. No vague adjectives: Vague adjectives have no definite meaning. They are considered a defect if they, similar to incomplete conditions, appear without any specification. E.g.: “The system should transmit data **quickly**.” Better: “The system should transmit data at a speed of **1000 MB/s**.”

3.2 Detection of Defects using linguistic rules: Two examples

We used the rule-based scripting language UIMA Ruta for the detection of defects. Prior to the scripting process, the requirements were run through a standard pipeline of natural language processing (NLP) modules. The pipeline separates the text into tokens (Tokenizer), maps the word-tokens to its part-of-speech tag (POS-Tagger) and groups words to larger grammatical units (Chunker). This information is needed by the rule script.

We will now present two examples to show how the patterns work. Since our work is based on German requirements and makes use of features of the German language, the examples will be given in German with an appropriate English translation followed by a literal translation to clarify the German word order.

No Use of Passive Voice

The passive voice in German is built with the auxiliary verb “werden” and the main verb in the past participle. Requirements describe something that is still supposed to happen or being built, therefore, a modal verb like “should” or “soll” in German is used in addition. The trick with this pattern is that there is no need to write a pattern for every possible combination of those three verbs; if they appear in a main sentence, the sequence “past participle” + “auxiliary verb” is sufficient to identify the passive voice. The following example, as seen in Figure 1, shows a German sentence that can be translated as “A QOS-Model must be made for the system”or word-for-word: “For the system must a QOS-Model made be.”

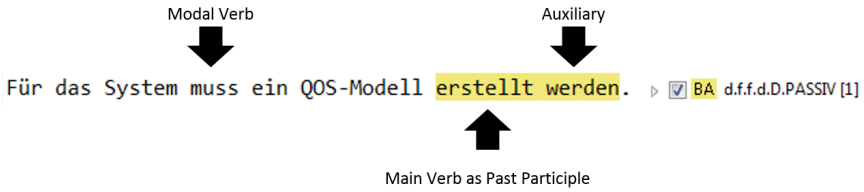

Figure 1: Annotation of Passive Voice in a German Sentence

No Vague Adjectives

The second example demonstrates the pattern used to identify vague adjectives. The challenge in this case was not only to identify certain adjectives, but also to determine whether there really was no further specification. The pattern first annotates a specification. We defined a specification as a noun phrase (an annotation delivered by the chunker) that holds a number or a unit. The information, which adjectives are seen as vague, differs between projects and is therefore stored in a list. Both sentences in 3.2 include a vague adjective – the German

word for “fast ” –, but only the first sentence was annotated because the adjective in the second case is preceded by a specification. The first sentence translates as “The system should transmit data quickly” or word-for-word “The System should fast transmit data.” The second translates as “The system should transmit data as quickly as 1000 MB/s.” or word-by-word: “The System should Data 1000 MB/s fast transmit.”

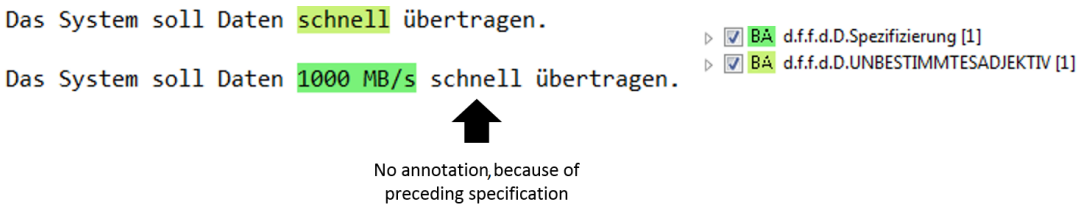

Figure 2: A vague adjective (‘schnell’) is only annotated if there is no specification.

4 Evaluation

To evaluate our approach, we randomly chose 100 requirements from a requirements stock that was written for a military project regarding information technology systems for command and control. An expert from our institute with a work experience of several years in the specified domain and in the assessment of requirements quality analyzed and annotated each requirement for the presence of a defect. Then the script was tested on the same requirements to evaluate how many of these manually annotated defects were found by the patterns, i.e. true positives (TP), how many were not found, i.e. false negatives (FN), and how often the patterns matched on something that wasn't a defect, i.e. false positives (FP). In total, a precision value ($\frac{TP}{TP+FP}$) of 73% and a recall value ($\frac{TP}{TP+FN}$) of 74% has been reached, which can be seen as a success since the patterns were written only on the basis of theory and without consultation of actual requirements. With 78 cases, Rule 9: ‘Be Atomic’ was the rule most commonly broken. We reached a precision of 0.742 and a recall of 0.846 with our patterns. Due to its many occurrences, this defect appears in different forms that had not been considered prior to the project, and thus were not found by the pattern. Another fairly common defect was Rule 2: ‘No empty verb phrases’, with 35 cases. Here, we reached a precision of 62.2% and a recall of 65.7%. This result can be improved, inter alia, by adding common phases like ‘es ermöglichen’ “realizing it” to the pattern. Some defect types occurred only rarely. There were only four cases of universal quantifiers, eight cases of indefinite adjectives and four cases of incomplete conditions. The best example of a working pattern has been the pattern to annotate passive forms. It found all seventeen cases, resulting in 100% precision and recall. This is a strong argument for the efficiency of the rule-based approach combined with specific linguistic knowledge.

5 Future Works and Lessons Learned

This project was designed to get an impression of whether rule-based detection of defects in German natural language requirements is a promising task, and the results suggest that it is. With some rather simple rules and without deep analysis of different requirements promising results have been achieved, that encourage us to work further in this field. The first task in the future will be to improve the rules based on the insight we gained from our evaluation.

Naturally, additional questions arose during the project, which should be addressed in the future: Apart from different linguistic forms that defects can appear in – the variation of natural language is never to be underestimated – it was interesting to see which defects correlate. The correlation between the different types of defects will also be an object of studies. Additionally, we aim to look into more and different requirement sets and identify common words that may cause such defects, which should increase the results for various rules.

References

[FFGL01] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, page 97. IEEE, 2001.

[FGR+18] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, pages 1–50, 2018.

[GFL+13] Gonzalo Génova, José M Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. A framework to measure and improve the quality of textual requirements. *Requirements engineering*, 18(1):25–41, 2013.

[Hue03] Claas Huesmann. Ein automat zur linguistischen analyse natürlichsprachlicher anforderungen, 2003.

[R+14] Christine Rupp et al. *Requirements-Engineering und-Management: Aus der Praxis von klassisch bis agil*. Carl Hanser Verlag GmbH Co KG, 2014.

[SBA13] Roxana Saavedra, Luciana C Ballejos, and Mariel Ale. Quality properties evaluation for software requirements specifications: An exploratory analysis. In *WER*, 2013.

[TB13] Sri Fatimah Tjong and Daniel M Berry. The design of sreea prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95. Springer, 2013.