Flexible Ambiguity Resolution and Incompleteness Detection in Requirements Descriptions via an Indicator-Based Configuration of Text Analysis Pipelines

Frederik S. Bäumer
University of Paderborn, Germany
fbaeumer@hni.upb.de

Michaela Geierhos
University of Paderborn, Germany
geierhos@hni.upb.de

Abstract

Natural language software requirements descriptions enable end users to formulate their wishes and expectations for a future software product without much prior knowledge in requirements engineering. However, these descriptions are susceptible to linguistic inaccuracies such as ambiguities and incompleteness that can harm the development process. There is a number of software solutions that can detect deficits in requirements descriptions and partially solve them, but they are often hard to use and not suitable for end users. For this reason, we develop a software system that helps end-users to create unambiguous and complete requirements descriptions by combining existing expert tools and controlling them using automatic compensation strategies. In order to recognize the necessity of individual compensation methods in the descriptions, we have developed linguistic indicators, which we present in this paper. Based on these indicators, the whole text analysis pipeline is ad-hoc configured and thus adapted to the individual circumstances of a requirements description.

1. Introduction

In software product management, one of the key challenges is the communication between stakeholders, especially end users, and software developers about desired software functions. These activities usually occur within the initiation phase of the project. This requirements analysis step is an investigation into the information and processing needs (i.e. functional and non-functional requirements) of the prospective end users. Current techniques for the requirements analysis step often use high-level language to precisely document user requirements, but NL makes it easier for (non-expert) stakeholders to participate in the requirements analysis process [1, 2]. Nevertheless, NL requirements descriptions may be considered as challenging because they are often inconsistent, ambiguous and incomplete [3, 4, 5, 6]. In particular, the occurrence of ambiguity and incompleteness is often discussed in literature [7]. There exist wide-ranging solutions (e.g. quality checklists, reading techniques, software tools) to assist stakeholders in unambiguously expressing and completing their individual software requirements. However, some methods such as quality checklists or reading techniques are more qualified for advanced use because they have to be trained first. In contrast, the aforementioned software tools should automatically point out mistakes in requirements descriptions, at least on paper. Existing tools are often designed for expert use only and are therefore limited to the detection of one or two specific deficits (e.g. lexical ambiguity). All solutions assume that all stakeholders are aware of possible quality deficits and are willing and even able to improve potential deficits in their NL requirements. We do not follow this assumption, because we do not necessarily ascribe the technical understanding of ambiguity and incompleteness resolution in software requirements to end users. This results in two research questions that we address in this work: (1) How can the detection and compensation of inaccuracies in requirements descriptions be made possible without user interaction? This fundamental question mainly concerns the state of research, which is currently dominated by identification approaches and hard-to-use special NL processing. (2) How do NL indicators have to be designed to allow a flexible and automatic compensation of inaccurate requirements descriptions?

To answer these research questions, we have to create text analysis pipelines, which take over the main optimization steps for end users. On the one hand, we have to select and combine appropriate (existing) tools for the detection and compensation of structural, referential and lexical ambiguity as well as for incompleteness in requirements descriptions. On the other hand, our goal is to automate the selection, control and coordination process of the necessary software modules as well as to implement an end-user-friendly

URI: http://hdl.handle.net/10125/50609
ISBN: 978-0-9981331-1-9

text optimization system for NL requirements. In this paper, we introduce quality features that trigger the ad-hoc configuration of the applicable text analysis pipeline for the detection and compensation of possible verbal deficits. Moreover, we want to provide insight into the development process of our prototype.

The structure of the paper is as follows: In Section 2, we give an overview of the related work, before we present the research methodology (cf. Section 3). Then we present the possible software modules of our requirements compensation pipeline (cf. Section 4), which is automatically configured as needed by applying the NL quality triggers defined in Section 5. We briefly present our current prototype in Section 6, then show and discuss our evaluation results in Section 7 before we finally conclude in Section 8.

2. Current State of Research

A lot of research has already been conducted in the field of software-assisted detection and compensation of deficits in NL requirements descriptions. In addition to linguistic expert solutions, which focus on a single linguistic inaccuracy phenomenon (e.g. lexical ambiguity), there are other approaches that can identify and resolve several deficits in requirements descriptions. Even mixed techniques exist that can detect different forms of ambiguity [8, 9] or, for example, ambiguity together with incompleteness [10, 11, 12]. An overview of existing approaches on disambiguation in the context of NL requirements is given by Husain and Beg [13] as well as Shah and Jinwala [14]. Shah and Jinwala [14] distinguish between different approaches based on pre-selected tools (e.g. Stanford Parser), chosen methods (e.g. rule-based, ontology-based, etc.) or the degree of automation. A further comprehensive overview of existing approaches on disambiguation in the requirements domain is given by Bano [15], which focuses on empirical work. All of these surveys show that a large number of existing software solutions only concentrate on the recognition of deficits. Consequently, the compensation remains the task of the stakeholders.

While Bajwa et al. [9] (NL2OCL) as well as Umber and Bajwa [4] (SR-Elicitor) focus on highly specialized approaches, Lami [16] (QuARS) and Bucchiarone et al. [17] (QuARS$_{express}$) present techniques that deal with a variety of verbal inaccuracies. But these methods also differ in their goals: While QuARS aims at detecting and highlighting as many deficits as possible in requirements descriptions, NL2OCL and SR-Elicitor concentrate on their detection and compensation. Furthermore, these tools do not expect user interaction in contrast to RESI [10, 18]. RESI takes requirements descriptions as graphs and checks them for linguistic deficits. If any problems are found, RESI will start a user dialog. In this case, not only the problematic texts are shown to the user, but also explicit compensation suggestions for each type of deficit. This requires linguistic resources that enable the rule-based detection of deficits on the one hand and provide additional information for the compensation on the other hand. Unfortunately, only few resources for NL requirements exist [19]. There is especially a lack of end-user requirements collections. Moreover, the compensation for incompleteness often relies on exhaustive resources, which are hard to get. Although there are compensation approaches for incomplete software requirements [20, 21], the underlying resources are still limited in scope.

Huertas and Juarez-Raminez [11, 22] introduce NLARE, a combined approach that concentrates on functional requirements and recognizes ambiguity, incompleteness and "atomism". The authors define ambiguity as gradable adjectives. NLARE detects incompleteness by using a matching-approach based on given questions ("Who", "What", "Where", "When"). Besides, "atomism" is an additional quality feature, which means that a single sentence only contains one requirement. For natural language processing (NLP), NLARE uses the Natural Language Toolkit (NLTK) and some regular expressions. In addition to the sentence boundary detection and tokenization, a spelling correction is included. As output, users get remarks such as "The requirement is ambiguous because it contains the word 'earlier' and 'later'" [11]. No further compensation hints or assistance is provided.

Just for the record, the above-mentioned methods have never been joined within a holistic approach for the improvement of requirements descriptions, although individual methods take account of several deficits: For example, the popular tool QuARS covers a whole range of ambiguities and other quality features (e.g. readability). However, a compensation does not take place, so high user interaction is at least necessary during the compensation step. User interaction is also a key factor influencing the performance and acceptance of the entire processing and is therefore considered as a possible bottleneck.

Stakeholders and especially end users expect their requirements to be entirely applied, but are not able to identify and compensate linguistic deficits in requirements descriptions on their own. A computer-aided but still interactive compensation process would be tedious. Furthermore, it cannot be ensured that end users detect any ambiguity or incompleteness even so it could be very frustrating. With this said, for example, NL2OCL would be an alternative that focuses on the compensation of lexical and syntactic ambiguity and does not rely on user interaction at all. However, NL2OCL needs additional structured information (e.g. class diagrams) for the disambiguation, which are most likely not available. Although there are approaches which can detect and/or compensate several forms of ambiguity and incompleteness, they do not fit stakeholder's needs, because they do not foresee user interaction and a high degree of automation. Actually, the link between the deficits in requirements descriptions and the appropriate compensation methods is still missing.

Here, we present our current work on indicators and strategies to combine existing NLP approaches to compensate linguistic deficits in NL requirements. We will show that our linguistic quality features are able to identify deficits in requirements descriptions and that they assist in configuring the compensation pipeline as needed. Thus, the selection of necessary compensation techniques no longer has to be carried out by the end user. Moreover, the number of further user inquiries is minimized, which is good for the performance and reduces the cognitive load of users.

Linguistic quality features for requirements descriptions are also known as "Requirements smells". They are mainly developed by Fermer et al. [23] and are very similar to the well known "Code smells", which are indicators for bad source code [24]. Fermer et al. [23] define requirements smell as "*an indicator of a quality violation, which may lead to a defect, with a concrete location and a concrete detection mechanism*".

3. Research Methodology

The research methodology in this paper follows the seven principles of design science reseach according to Hevner et al. [25], which serve as the basis for the design of an IT artefact. Therefore, design science research must deliver an IT artefact as a result (*design as an artifact*). Here, we deliver a software tool as IT artefact. Furthermore, we provide a methodical contribution to the software-supported improvement of user-defined requirements descriptions by dealing with ambiguity and incompleteness. This is also determined by further design science principles (*research contribution, problem relevance*). An IT artifact must always be related to a problem and contribute to the research field. In our case, we tackle the problem of the inaccurate requirements description and make a contribution to their automatic correction. Therefore, an IT artifact has to be evaluated with regard to the benefit (*design evaluation*). For this reason, we evaluate our developed quality indicators on real requirements descriptions.

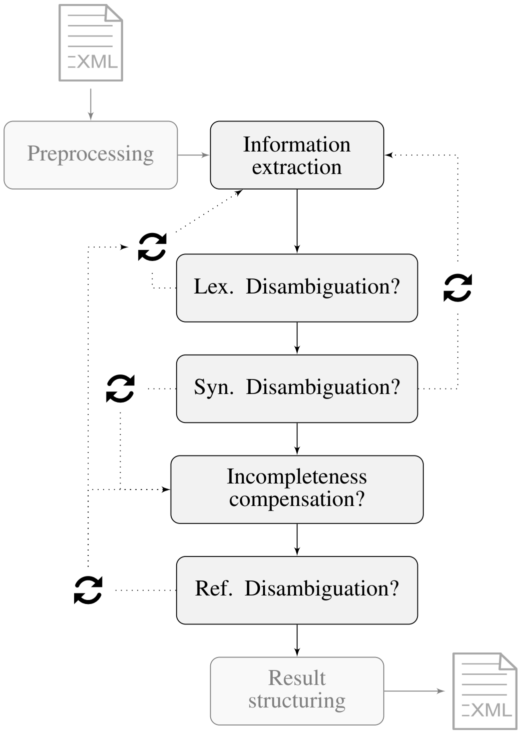


Figure 1. Sample configuration of an indicator-based text analysis pipeline

Since design science must work methodically in the design as well as in the evaluation of the artifact (*research rigor*), we see our contribution as a response to the existing state of research. Although, we proceed until we find a suitable solution (*design as a search process*), only the final solution is presented in this paper. In addition to the IT artifact itself, the results are also comprehensible for a non-technical audience (*communication of research*).

4. Requirements Processing Pipeline

The goal of our system is the automatic detection and compensation of ambiguity and incompleteness in NL software requirements. For this purpose, existing disambiguation and completion techniques are jointly applied on a requirements description if necessary.

Our system configures the adequate text analysis pipeline and interprets the resolution results, before the optimized requirements description is shown to the end users. In this regard, the identification of suitable detection and compensation methods for lexical, syntactic and referential ambiguity, as well as incompleteness in NL requirements is a preliminary, but crucial step. The configuration, execution and

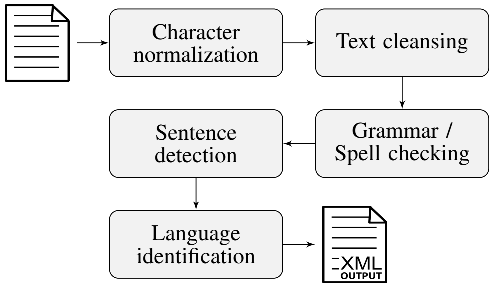


Figure 2. Preprocessing workflow

monitoring of the resulting NLP pipeline is done by a so-called compensation strategy, which allows to reconcile results and benefits from some synergy effects between the applied methods. That is, for example, semantic information can be obtained from syntactic disambiguation.

In this paper, we pursue an automatic strategy that is self-configuring and embedded in a predefined processing context. For this purpose, with high performance in mind, the development of context-sensitive indicators is necessary for ad-hoc configuration of an NLP pipeline that can handle individual deficits in requirements descriptions (cf. Figure 1). Before applying this strategy, text preprocessing and information extraction are conducted in order to filter requirement-related statements (so-called on-topic information).

In other words, the text analysis pipeline includes a preprocessing step (cf. Figure 2) that is responsible for the overall text-quality improvement. But it also provides valuable information that the indicators need for NLP pipeline configuration (e.g. whether a sentence is on- or off-topic). The on-/off-topic classification is done by REaCT [26], which is also used for information extraction purposes. REaCT is able to recognize main semantic information bits such as "role" or "action" (so-called process words) in requirements descriptions. The enrichment with semantic information is crucial for the whole process because the compensation of incompleteness relies on the structured output produced by this task. Furthermore, each module such as the referential disambiguation is triggered by its respective requirements quality indicator.

5. NL Requirements Quality Indicators

How can imprecise statements be detected? We can test for quality violations in requirements descriptions by applying so-called linguistic triggers that can spot ambiguous or missing information bits. These indicators emerge at a certain passage in a text and trigger at least one detection and compensation module (e.g. incompleteness compensation). In the following, the already-implemented triggers are presented, starting with the indicators for different types of ambiguity, followed by the indicator for incompleteness.

5.1. Lexical Ambiguity Trigger

The lexical disambiguation aims at assigning the correct sense to a lexeme from a set of possible readings (after contextualization). For instance, lexical ambiguity for a given token is triggered when more than one reading is probable. But is it really necessary to check all the lexemes of any requirements description for lexical ambiguity? And are there any constraints that could minimize the set of potential readings? Hence, the following restrictions were made before applying this trigger: Only tokens that are embedded in on-topic sentences and have some semantic function within a requirement (e.g. an action) are considered for disambiguation in order to increase efficiency. This way, the only non-stopwords that are disambiguated are those that have not been classified as the semantic category "role" or "priority".

Given the sample NL requirement "*I want to send emails to my family*" in Figure 3. In step (1), the original sentence is kept, which is labeled with additional semantic information in step (2) through REaCT [26]. Stopwords that have no semantic function within an NL requirement (in this case "to") are removed in step (3). The same is done for lexemes categorized as "role" or "priority" information. Subsequently, the remaining lexemes are POS-annotated (through part-of-speech tagging) in step (4). Here, further stopwords, based on the recognized POS tags, are removed. Finally, step (5) shows the remaining disambiguation candidates: "send", "emails" and "family". This pre-selection allows a look-up in a taxonomy for English such as WordNet, which contains nouns, verbs, adjectives and other word classes, grouped into so-called "synsets". By means of WordNet, we can identify all senses for a given lexeme with its corresponding POS tag. The lexemes "send" (as verb) and "family" (as noun) both have eight different readings according to WordNet. In contrast, "emails" has only one sense and therefore is excluded from the disambiguation candidate list. For this example, the indicator check tells us that two out of eight lexemes are potentially ambiguous.

On the one hand, we have to decide whether all possible readings according to WordNet are relevant, or if further restrictions have to be made. On the other hand, we have to clarify how reliable the

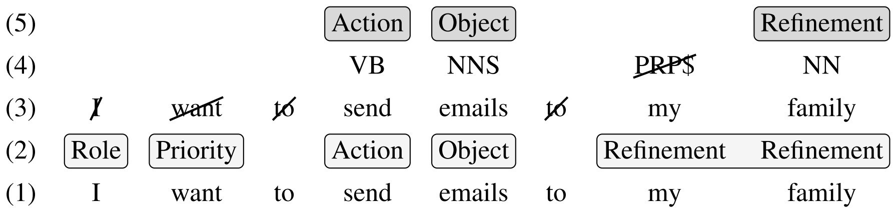

(5) Action Object Refinement
(4) VB NNS ~~PRP$~~ NN
(3) ~~I~~ ~~want~~ ~~to~~ send emails ~~to~~ my family
(2) Role Priority Action Object Refinement Refinement
(1) I want to send emails to my family

Figure 3. Triggered lexical ambiguity

indicators perform in requirements descriptions: Is the suspicion of ambiguity for two tokens suf-ficient to apply lexical disambiguation in the NLP pipeline? Since the underlying candidates have already been prefiltered and are definitely part of the core statements (e.g. "send" = action), it is sufficient to know that at least one of these lexemes is ambiguous and that it could lead to misinpretations in further processing steps. The task of indicators is not to conduct disambiguation but to trigger its integration into the respective NLP pipeline. For now, it will suffice to store the information that several readings are available and that a given lexeme is considered as ambiguous.

5.2. Syntactical Ambiguity Trigger

In the following, we describe how to spot coordination and prepositional phrase (PP) binding ambiguity. Since these types of syntactic ambiguities are structural phenomena, syntactic patterns are needed as indicators.

Indicator for coordination ambiguity. Here, conjunctions and syntactic patterns are used as triggers. On the one hand, coordination ambiguity occurs when a modifier ("JJ") refers to coordinated nouns. Then it is not clear if only the noun ("NNS") before the conjunction ("CC") or even the one after is modified (cf. example A). This kind of coordination ambiguity can be detected by hand-crafted patterns (e.g. "JJ NNS CC NNS"). On the other hand, it appears when concatenating nouns with conjunctions (cf. example B). In both cases, the existence of conjunctions (e.g. "*and*" and "*or*" [27, 28]) is crucial and is checked before patterns are applied on each sentence.

(**A**)
I use crawlers and spiders and users report me
PRP VBP NNS **CC** NNS **CC** NNS VBP PRP

(**B**)
I want to send large emails and tasks
PRP VBP TO VB **JJ** NNS **CC** NNS

As shown in the sample requirements descriptions A and B, both types of the coordination ambiguity are triggered by at least one conjunction (cf. example A). Therefore, this is used as a preselection criterion. Following this, sentences containing at least one conjunction are checked for further conjunctions. The indicator detects potential coordination ambiguity if at least two conjunctions are given (cf. example B).

Indicator for PP binding ambiguity. Here, we adapt existing syntactic patterns for detecting potential ambiguity, such as "V NP PP" [29, 30]. This pattern, for example, identifies prepositional phrases ("PP") succeeding nominal phrases ("NP") in object position. The necessary syntactic information can be generated by Shallow Parsing approaches, which are considered to be very performant and reliable [31]. However, the question arises whether this pattern can be further restricted in terms of high performance. One restriction could be, for example, the exclusion of certain prepositions that are not regarded as (highly) ambiguous. According to an English dictionary, there exist many prepositions[1]. The most common preposition "of" is also one that is not considered to be ambiguous because it often represents a genitive and is therefore bound to an NP. In almost all cases, it can be excluded as a disambiguation candidate [34]. Therefore, "of" is ignored as an ambiguous preposition ("PREP") within this work. For this reason, we added the restriction that a PP does not have to be introduced by "of" ("V NP PP" | PREP $\neq$ "of").

5.3. Referential Ambiguity Trigger

A possible approach for the detection of referential ambiguity would be to check whether pronouns are given in requirements descriptions. Since pronouns are also called "substitutes" of nouns (antecedents) [35], they might be assigned to the wrong antecedent [36]. This test can be conducted by POS tagging or by means of word lists. One restriction, to ensure

[1] DELA contains 124 prepositions, Davies [32] lists 196 and Essberger's list [33] contains 150 with the reference that there are many more prepositions.

performance, is to concentrate only on high-frequent pronouns. However, pronouns have not proved a recipe for referential ambiguity detection. For this reason, a trigger is necessary, that considers both contextual and semantic information.

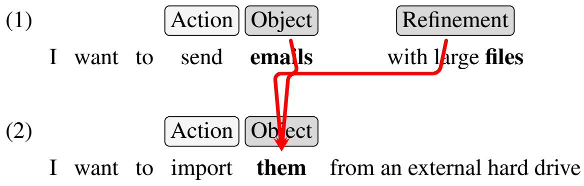

(1)
Action
Object
Refinement
I want to send emails with large files
(2)
Action
Object
I want to import them from an external hard drive

Figure 4. Triggered referential ambiguity

The example, shown in Figure 4, contains an extract of a sample requirements description consisting of two consecutive sentences. In sentence (2) the personal pronoun “them” is used as object, whereas it remains unclear what its antecedent is: It could be the object or the semantic category of the refinement in sentence (1) because both nouns are plural. This illustrates that indicators for referential ambiguity should be determined within a whole discourse because antecedent and direct anaphoric reference can occur in the same or in consecutive sentences. In Figure 4, the pattern “NNS + NNS + them” is sufficient for disambiguation. It can be extended by considering antecedents in singular, as in “*I want to send an email with an attachment. It is a very large one.*”. Substituting “them” by all possible POS tags for pronouns, the new pattern (“NN (S) + NN (S) + PRP”) covers more occurrences of referential ambiguity. However, the pattern works only if either two antecedents occur in the same sentence together with a pronoun or in the preceding sentence. Here, it is assumed that the last-mentioned antecedent is usually meant.

5.4. Incompleteness Trigger

We define incomplete requirements descriptions as named software requirements that are imprecise because of missing information bits (so-called partially incomplete requirements). Therefore, our developed trigger considers existing information to draw conclusions on how to fill in the missing slots. For incompleteness detection, we pursue the approach by Bäumer and Geierhos [20], which mainly relies on Semantic Role Labeling (SRL) and on a fine-grained analysis of the Predicate Argument Structure (PAS) of requirements. This detection and compensation of incompleteness is difficult because the lack of data has to be examined. For this reason, the existing semantic categories, which were annotated by REaCT, are essential clues for the incompleteness trigger.

As the first indicator for incompleteness, the semantic category “action” will be investigated. Since process words are the content words of a requirement, they cannot be reliably compensated without context information. We therefore treat a sentence without process words as off-topic and ignore it in later processing steps. In most cases, the off-topic classification has already labeled requirements without process words as irrelevant during the preprocessing step.

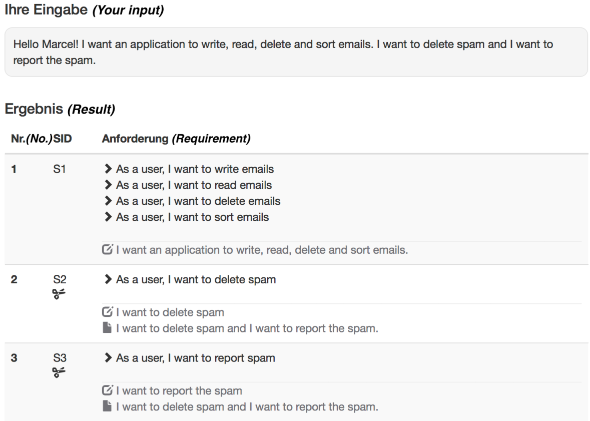
In addition to process words, the semantic categories for “role”, “component”, and “object” are taken into account by this indicator. These categories usually represent the arguments of a predicate used in requirements descriptions. In general, the subject of an NL requirement is occupied by the role or component, although it is sufficient to find one of the two semantic categories in a sentence. Since incompleteness can be assumed for a missing subject, the indicator adds the module for incompleteness compensation to the text analysis pipeline. However, the lack of the subject could have already been identified during the requirements extraction steps (by REaCT), especially when end users provided a list of process words, as the following example shows: “**I**$_{\text{Role}}$ want to **write**$_{\text{Action}}$, read$_{\text{Action}}$ and **send**$_{\text{Action}}$ e-mails$_{\text{Object}}$”. Here, the requirement extraction correctly assigns the subject (“I”) to the predicate “write”, but ignores it as an argument for “send”.

Another argument of the predicate can be filled by the semantic category “object”, such as “e-mails” in the given example. In contrast to the subject and the predicate, the object is not required to make up a well-formed sentence. However, it is often necessary to create a meaningful and even precise sentence. Additionally, it is assumed that an object is an obligatory requirements description if the process word is expressed by a transitive verb. Therefore, a requirement is considered as incomplete if the semantic category “object” is missing.

6. Prototype

We implemented the developed triggers (Section 5) for the automatic compensation strategy by including all necessary compensation modules in a prototype in order to test the functionality of our concept. Our software system allows end users to formulate their own requirements for a desired software in NL and automatically checks the descriptions for various errors and weaknesses (i.e. ambiguity and incompleteness) and compensates them.

The idea is that end users write their software requirements in NL and send them to our system (via a provided web interface). Then, the system conducts preprocessing and applies the aforementioned indicators in order to configure the automatic compensation strategy. This strategy covers not only the compensation of ambiguity and incompleteness but also the transformation of the detected functional requirements into a structured output (cf. Figure 5). Figure 5 shows that even off-topic information is filtered during the preprocessing step and complex sentences are simplified to guarantee a more robust processing (English translations are not part of the user interface of the prototype). The output is given in simplified language, which provides the end user a quick overview to check if all functional requirements have been recognized by the system. Here, two perspectives are supported: the user's view (e.g. user, administrator) and the system's view (e.g. application, system). The controlled syntax for the user view is, for example, defined as "As a *<role>*, *<pronoun>* *<priority>* *<action>* *<object>*" and is based on Dollmann's information extraction template [37].



Ihre Eingabe ***(Your input)***

Hello Marcel! I want an application to write, read, delete and sort emails. I want to delete spam and I want to report the spam.

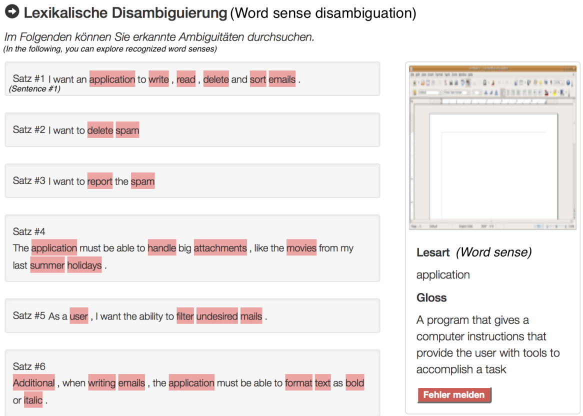
Ergebnis ***(Result)***

Nr.*(No.)*	**SID**	**Anforderung *(Requirement)***
1	S1	› As a user, I want to write emails › As a user, I want to read emails › As a user, I want to delete emails › As a user, I want to sort emails I want an application to write, read, delete and sort emails.
2	S2	› As a user, I want to delete spam I want to delete spam I want to delete spam and I want to report the spam.
3	S3	› As a user, I want to report spam I want to report the spam I want to delete spam and I want to report the spam.

Figure 5. Results (semi-structured requirements)

However, this is a simplified view for end users to understand the processing result and, if necessary, to correct it. There are also additional user interfaces which, for example, explain the applied word sense disambiguation (cf. Figure 6) or explain which quality indicators were found in the provided requirements description. These additional interfaces are necessary for debugging. An example for the word sense disambiguation is the token "colleagues", which can also mean "Woollahra Colleagues Rugby Football Club" within the category "Rugby union teams in Sydney". Furthermore, an wrong disambiguation can occur through incorrect POS tags. Through the visualization, users can recognize errors and finally correct them manually, even if this should affect a minimum of cases.

Moreover, the system has an XML interface that provides the (on-topic) functional requirements as well as the results of the various disambiguation and compensation steps for further processing purposes.



Lexikalische Disambiguierung (Word sense disambiguation)

Im Folgenden können Sie erkannte Ambiguitäten durchsuchen.
(In the following, you can explore recognized word senses)

Satz #1 I want an application to write , read , delete and sort emails .
(Sentence #1)

Satz #2 I want to delete spam

Satz #3 I want to report the spam

Satz #4
The application must be able to handle big attachments , like the movies from my last summer holidays .

Satz #5 As a user , I want the ability to filter undesired mails .

Satz #6
Additional , when writing emails , the application must be able to format text as bold or italic .

Lesart *(Word sense)*

application

Gloss

A program that gives a computer instructions that provide the user with tools to accomplish a task

Figure 6. Result explanation (lexical ambiguity)

7. Evaluation

The indicators described in Section 5 rely on linguistic features and patterns determined by different tools and resources (e.g. POS tagger, WordNet). Individual tools as well as the interaction of several modules can lead to errors (incorrect or undetected indicators). A high indicator reliability is achieved if the detected necessity of a compensation is predominantly correct. Then the underlying tools have a low error rate for the previously defined features and patterns.

7.1. Results

In the following, it is necessary to evaluate how often errors occur during the indicator detection. This can be understood as a binary classification problem: Either features (e.g. for lexical ambiguity) are or are not given in a requirements description. The accuracy of the indicators could be determined by how many requirements descriptions were correctly classified in relation to all descriptions:

$$a = \frac{\text{correct classified descriptions}}{\text{all checked descriptions}} \qquad (1)$$

Conversely, this means that the error rate f can be described by $f = 1 - a$. However, this measure of the accuracy or error rate is to be regarded as superficial, since there are four possible result combinations (true positive, true negative, false positive, false negative), of which up to now only two are considered (true positives and negatives).

However, it is not always easy for human readers to identify characteristics reliably, so three (guided) evaluators evaluate the requirements. The result of the software system is then compared with the joint result (majority decision) of the evaluators. Based on 400 randomly selected requirement descriptions from our requirement corpus, Table 1 represents the result combinations for each indicator.

	TP	***TN***	***FP***	***FN***
INC	171	120	34	75
REF	156	170	11	63
SYN	142	179	22	57
WSD	400	0	0	0

Table 1. Frequency of result combinations

As noted before, a “global value is not sufficient for the [...] accuracy” [31], since how reliably the methods work is also of interest. At this point, two established evaluation measures are used: Recall (r) and Precision (p).

$$r = \frac{TP}{TP + FN} \tag{2}$$

$$p = \frac{TP}{TP + FP} \tag{3}$$

In order to increase the meaningfulness of the evaluation values, the harmonic mean is used in the following ($\beta = 1$).

$$F_\beta = (1 + \beta^2) \cdot \frac{p \cdot r}{(\beta^2 \cdot p) + r} \tag{4}$$

At this point, it should be kept in mind that each compensation method added to the compensation pipeline due to incorrect indicator decisions (equivalent to false positives) has only a negative influence on the total running time, whereas the non-consideration of an indicator (missing compensation) leads to a deterioration in the result (*false negatives*) because the necessary compensation does not take place. As shown in Table 1, this currently affects several indicators. At this point, it is recommended to modify the F-score with respect to the higher weighting of the Recall to take the influence of false negatives more into account ($\beta = 2$). Table 2 shows the results of the different evaluation measures.

7.2. Discussion

The evaluation results show that the indicators work in principle. However, it also shows that errors can occur, which can lead to insufficient processing pipelines. In particular, Table 2 shows that the compensation of incompleteness (INC) has the lowest F$_2$-score, which is due to a low recall. In contrast, referential disambiguation (REF) is equivalent to incompleteness with regard to the recall, but has a much higher precision ($\Delta\ 0,1$). This has only a low effect on the F$_2$-score ($\Delta\ 0,03$). At this point, a difference in the accuracy can be detected ($\Delta\ 0,09$). The syntactic disambiguation (SYN) has a high precision and a good recall. Only the lexical disambiguation appears to be more accurate, but this is not necessarily true: Lexical ambiguity can be determined on the basis of individual tokens, but the indicators attest ambiguity and incompleteness for whole requirements descriptions. It is therefore very likely that at least one ambiguous token is correctly detected as ambiguous in all descriptions and thus both recall and precision are very good. Of course there are also errors in the WSD indicator application: Errors are caused, for example, by missing entries in the underlying resources. The WSD indicator thus represents a special case.

As a constraint of this evaluation, it must be stated that the requirements descriptions are real-life specifications that are limited to a few sentences each. This is mainly due to the fact that more extensive requirements are not accessible [19]. Our collection of NL requirements is, to the best of our knowledge, one of the largest available [38, 39]. However, the design of our prototype should be scalable to enable the processing of requirements of complex real-life systems. The underlying design of the prototype is not limited in the scope of the requirements. In the case of very extensive requirements, performance problems of the individual components may occur. This has to be evaluated in future research. Furthermore, domain-specific characteristics such as vocabulary is conceivable, which is not covered by currently used linguistic resources. Moreover, list of requirements which can not be classified into a syntactic structure, are conceivable and currently supported only rudimentarily. In future work, we want to enhance the robustness of our indicators and continue resource development.

8. Conclusion

Within this work, we provide a methodical contribution to the software-supported improvement of user-defined requirements descriptions by dealing with

	Accuracy	***Recall***	***Precision***	***F_1-Score***	***F_2-Score***
INC	0.73	0.70	0.83	0.76	**0.72**
REF	0.82	0.71	0.93	0.81	**0.75**
SYN	0.80	0.71	0.87	0.78	**0.74**
WSD	1.00	1.00	1.00	1.00	**1.00**

Table 2. Evaluation results of the indicator quality

ambiguity and incompleteness. We were able to show that the detection and compensation of inaccuracies in requirements descriptions can be made possible without user interaction (Research question 1) by developing a flexible processing pipeline. Therefore, we presented our data-driven linguistic indicators that allow to optimize the common text analysis pipeline by means of a needs-oriented analysis. The ad-hoc configuration of the compensation pipeline minimizes user interaction as well as program runtime. We were able to show that rule-based indicators already cover the linguistic phenomena well and that existing resources can be used (Research question 2). This procedure is reliable and performant, especially because the actual compensating components are not used in the indicator stage.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901).

References

[1] M. Geierhos, S. Schulze, and F. S. Bäumer, “What did you mean? Facing the Challenges of User-generated Software Requirements,” in *Proceedings of the 7th ICAART* (S. Loiseau, J. Filipe, B. Duval, and J. van den Herik, eds.), Special Session on PUaNLP 2015, (Lissabon, Portugal), pp. 277–283, SCITEPRESS – Science and Technology Publications, 2015.

[2] A. Ferrari, F. dell’ Orletta, G. O. Spagnolo, and S. Gnesi, “Measuring and Improving the Completeness of Natural Language Requirements,” in *Requirements Engineering: Foundation for Software Quality* (C. Salinesi and I. van de Weerd, eds.), vol. 8396 of *LNCS*, pp. 23–38, Essen, Germany: Springer, 2014.

[3] V. Pekar, M. Felderer, and R. Breu, “Improvement Methods for Software Requirement Specifications: A Mapping Study,” in *Proceedings of the 9th QUATIC*, (Guimarães, Portugal), pp. 242–245, IEEE, Sept. 2014.

[4] A. Umber and I. S. Bajwa, “Minimizing Ambiguity in Natural Language Software Requirements Specification,” in *Proceedings of the 6th ICDIM*, (Melbourn, Australia), pp. 102–107, IEEE, Sept. 2011.

[5] E. Kamsties, “Understanding Ambiguity in Requirements Engineering,” in *Engineering and Managing Software Requirements* (A. Aurum and C. Wohlin, eds.), pp. 245–266, Berlin / Heidelberg, Germany: Springer, 2005.

[6] M. Osborne and C. K. MacNish, “Processing Natural Language Software Requirement Specifications,” in *Proceedings of the 2nd International Conference on Requirements Engineering*, (Colorado Springs, CO, USA), pp. 229–236, IEEE, Apr. 1996.

[7] QRA Corp, “Leveraging natural language processing in requirements analysis: How to eliminate over half of all design errors before they occur.” IEEE Spectrum Whitepaper, 2017.

[8] S. F. Tjong and D. M. Berry, “The Design of SREE – A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned,” in *Requirements Engineering: Foundation for Software Quality* (J. Doerr and A. L. Opdahl, eds.), vol. 7830 of *LNCS*, pp. 80–95, Berlin / Heidelberg, Germany: Springer, 2013.

[9] I. S. Bajwa, M. Lee, and B. Bordbar, “Resolving Syntactic Ambiguities in Natural Language Specification of Constraints,” in *Computational Linguistics and Intelligent Text Processing* (A. Gelbukh, ed.), vol. 7181 of *LNCS*, pp. 178–187, Berlin / Heidelberg, Germany: Springer, 2012.

[10] S. J. Körner, *RECAA - Werkzeugunterstützung in der Anforderungserhebung*. PhD thesis, KIT, Karlsruhe, Germany, Feb. 2014.

[11] 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*, CUBE ’12, (New York, NY, USA), pp. 371–378, ACM, 2012.

[12] F. Fabbrini, M. Fusani, S. Gnesi, and G. 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*, (Greenbelt, MD, USA), pp. 97–105, Nov. 2001.

[13] S. Husain and R. Beg, “Advances in Ambiguity less NL SRS: A review,” in *Proceedings of ICETECH 2015*, (Coimbatore, TN, India), pp. 221–225, Mar. 2015.

[14] U. S. Shah and D. C. Jinwala, “Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey,” *SIGSOFT Software Engineering Notes*, vol. 40, pp. 1–7, Sept. 2015.

[15] M. Bano, “Addressing the Challenges of Requirements Ambiguity: A Review of Empirical Literature,” in *Proceedings of the 5th International Workshop on EmpiRE*, (Ottawa, ON, Canada), pp. 21–24, IEEE, August 2015.

[16] G. Lami, “QuARS: A Tool for Analyzing Requirements,” Technical Report ESC-TR-2005-014, Carnegie Mellon University, Sept. 2005.

[17] A. Bucchiarone, S. Gnesi, A. Fantechi, and G. Trentanni, “An Experience in Using a Tool for Evaluating a Large Set of Natural Language Requirements,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC ’10, (New York, NY, USA), pp. 281–286, ACM, 2010.

[18] S. J. Körner and T. Brumm, "Natural Language Specification Improvement with Ontologies," *International Journal of Semantic Computing*, vol. 03, no. 04, pp. 445–470, 2010.

[19] W. F. Tichy, M. Landhäußer, and S. J. Körner, "nlrpBENCH: A Benchmark for Natural Language Requirements Processing," in *Multikonferenz Software Engineering & Management 2015*, Mar. 2015.

[20] F. S. Bäumer and M. Geierhos, "Running out of Words: How Similar User Stories Can Help To Elaborate Individual Natural Language Requirement Descriptions," in *Proceedings of the ICIST 2016* (G. Dregvaite and R. Damasevicius, eds.), CCIS, (Druskininkai, Lithuania), pp. 549–558, Springer, Oct. 2016.

[21] M. Geierhos and F. S. Bäumer, "How to Complete Customer Requirements: Using Concept Expansion for Requirement Refinement," in *Proceedings of the 21st NLDB* (E. Métais, F. Meziane, M. Saraee, V. Sugumaran, and S. Vadera, eds.), (Manchester, UK), June 2016.

[22] C. Huertas and R. Juárez-Ramírez, "Towards assessing the quality of functional requirements using english/spanish controlled languages and context free grammar," in *Proceedings of the 3rd International Conference on DICTAP*, pp. 234–241, SDIWC, July 2013.

[23] H. Femmer, D. M. Fernndez, S. Wagner, and S. Eder, "Rapid Quality Assurance with Requirements Smells," *Journal of Systems and Software*, 2016. In Press, Corrected Proof. See: `http://www4.in.tum.de/~femmer/works/2016-requirements_smells-jss.pdf`. Accessed: 27.08.2016.

[24] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Object Technology Series, Addison-Wesley, 1999.

[25] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, pp. 75–105, Mar. 2004.

[26] M. Dollmann and M. Geierhos, "On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements," in *Proceedings of the Conference on EMNLP*, Nov. 2016.

[27] D. M. Berry, E. Kamsties, and M. M. Krieger, "From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity – A Handbook. Version 1.0." `https://cs.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf`, 2003. Accessed: 16.11.2015.

[28] F. Chantree, A. Kilgarriff, A. De Roeck, and A. Willis, "Disambiguating Coordinations Using Word Distribution Information," in *Proceedings of Recent Advances in Natural Language Processing*, (Borovets, Bulgaria), Sept. 2005.

[29] E. Agirre, T. Baldwin, and D. Martinez, "Improving Parsing and PP attachment Performance with Sense Information," in *Proceedings of the Annual Meeting of the ACL*, (Columbus, OH, USA), pp. 317–325, ACL, June 2008.

[30] K. Nadh and C. Huyck, "Prepositional Phrase Attachment Ambiguity Resolution Using Semantic Hierarchies," in *The 9th International Conference on Artificial Intelligence and Applications*, (Innsbruck, Austria), ACTA Press, Jan. 2009.

[31] K.-U. Carstensen, C. Ebert, C. Ebert, S. Jekat, R. Klabunde, and H. Langer, eds., *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Heidelberg: Spektrum Akademischer Verlag, 3 ed., 2010.

[32] M. Davies, "Word frequency data – Corpus of Contemporary American English." `http://www.wordfrequency.info/free.asp?s=y`, 2016. Accessed: 08.09.2016.

[33] J. Essberger, *English Prepositions List – 150 Prepositions*. Cambridge, UK: Englishclub.com, 2012.

[34] A. Ratnaparkhi, "Statistical Models for Unsupervised Prepositional Phrase Attachment," in *Proceedings of the 17th COLING - Volume 2*, COLING '98, (Stroudsburg, PA, USA), pp. 1079–1085, ACL, 1998.

[35] J. Dittmann and R. Thieroff, *Richtiges Deutsch leicht gemacht*. Bertelsmann Wahrig, Gütersloh / München: Wissenmedia, 2009.

[36] IEEE, *ISO/IEC/IEEE 29148 – Systems and software engineering – Life cycle processes – Requirements engineering*. New York, NY, USA: IEEE, Dec. 2011. ISO/IEC/IEEE 29148:2011(E).

[37] M. Dollmann, "Frag die Anwender: Extraktion und Klassifikation von funktionalen Anforderungen aus User-Generated-Content," master thesis, Paderborn University, Paderborn, Mar. 2016.

[38] F. S. Bäumer, *Indikatorbasierte Erkennung und Kompensation von ungenauen und unvollständig beschriebenen Softwareanforderungen*. PhD thesis, University of Paderborn, 2017.

[39] F. S. Bäumer, M. Dollmann, and M. Geierhos, "Studying software descriptions in sourceforge and app stores for a better understanding of real-life requirements," in *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics* (F. Sarro, E. Shihab, M. Nagappan, M. C. Platenius, and D. Kaimann, eds.), pp. 19–25, ACM, September 2017.