Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain

Benedetta Rosadini[1], Alessio Ferrari[3](✉), Gloria Gori[2], Alessandro Fantechi[2],
Stefania Gnesi[3], Iacopo Trotta[1], and Stefano Bacherini[1]

[1] Alstom Ferroviaria S.p.A., Florence, Italy
{benedetta.rosadini,iacopo.trotta,
stefano.bacherini}@transport.alstom.com
[2] University of Florence, DINFO, Florence, Italy
{gloria.gori,alessandro.fantechi}@unifi.it
[3] ISTI-CNR, Pisa, Italy
{alessio.ferrari,stefania.gnesi}@isti.cnr.it

Abstract. *Context and motivation*: In the railway safety-critical domain requirements documents have to abide to strict quality criteria. Rule-based natural language processing (NLP) techniques have been developed to automatically identify quality defects in natural language requirements. However, the literature is lacking empirical studies on the application of these techniques in industrial settings. *Question/problem*: Our goal is to investigate to which extent NLP can be practically applied to detect defects in the requirements documents of a railway signalling manufacturer. *Principal idea/results*: To address this goal, we first identified a set of typical defects classes, and, for each class, an engineer of the company implemented a set of defect-detection patterns by means of the GATE tool for text processing. After a preliminary analysis, we applied the patterns to a large set of 1866 requirements previously annotated for defects. The output of the patterns was further inspected by two domain experts to check the false positive cases. *Contribution*: This is one of the first works in which defect detection NLP techniques are applied on a very large set of industrial requirements annotated by domain experts. We contribute with a comparison between traditional manual techniques used in industry for requirements analysis, and analysis performed with NLP. Our experience tells that several discrepancies can be observed between the two approaches. The analysis of the discrepancies offers hints to improve the capabilities of NLP techniques with *company specific* solutions, and suggests that also company practices need to be modified to effectively exploit NLP tools.

Keywords: NLP · Requirements · Ambiguity · Defect detection · Quality

1 Introduction

The CENELEC norms provide standards for the development of railway safety-critical systems in Europe. The CENELEC EN 50128:2011 [6], specific for

© Springer International Publishing AG 2017
P. Grünbacher and A. Perini (Eds.): REFSQ 2017, LNCS 10153, pp. 344–360, 2017.
DOI: 10.1007/978-3-319-54045-0_24

software, asks requirements documents for railway systems to be *complete, clear, precise, unequivocal, verifiable, testable, maintainable, and feasible.* To ensure that these quality attributes are met, companies developing railway products have a Verification Engineer (VE) who reviews for defects any requirements document produced along the development process. This review activity is time consuming and error prone, and an automated review assistant might help VEs in their task. As well known, requirements are normally edited in natural language (NL) [19], and the railway domain makes no exception. Several natural language processing (NLP) approaches have been developed to assist requirements review. Part of these works focuses on the identification of typical defective terms and constructions [2,4,11,14,15,20], while other focus on artificial intelligence techniques [7,13,21]. However, the literature is lacking large-scale case studies concerning industrial applications of NLP approaches for defect detection [11]. This papers aims at filling this research gap, by providing the experience done within a collaboration between a world-leading railway signalling company, the University of Florence, and ISTI-CNR to investigate the feasibility of using NLP for defect identification in the requirements documents of the company. This experience, which involved three professional VEs and a large-scale experimentation on 1866 requirements, shows that NLP technologies can be used to develop *in-house* tools for defect identification. The internal development of the tools can enable the VEs of the company to tune the tools to account for part of the discrepancies that occur between manual reviews and automated ones.

The remainder of the paper is structured as follows. Section 2 summarises related works. In Sect. 3, we provide an overview of the current work. In Sect. 4 we describe the patterns adopted for defect detection. Sections 5 and 6 provide the results of a preliminary and a large-scale study, respectively, on the application of the patterns. In Sect. 7 we provide an analysis of the false positive cases performed on the large-scale study. Section 8 highlights the lessons learned.

2 Related Works

The literature counts several contributions concerning the application of NLP techniques to detect defects in NL requirements. These works can be categorised into those that use rule-based approaches [2,4,11,14,15,20] and those that leverage artificial intelligence approaches [7,13,21]. Our contribution falls into the first category, which collects all the works in which defects are identified based on linguistic patterns. Hence, we briefly discuss relevant works in this category.

The Ambiguity Handbook of Berry *et al.* [4] includes one of the most influential classification of ambiguity-related defects in requirements, and provides a large set of examples of typically dangerous words and constructions. Gnesi *et al.* [15] present QuARS, a tool for defect detection based on a quality model developed by the authors. Similarly, Gleich *et al.* [14] implemented a grep-like, pattern-based technique to detect defects, supported by statistical NLP techniques such as POS tagging. Tjong and Berry [20] developed SREE, a tool that identifies defects based on a pre-defined list of dangerous terms. Arora *et al.* [2] use patterns of linguistic defects as the other works, and, in addition, checks the conformance of the requirements to a given template. All these works were used as fundamental references to define the defect detection patterns of our study. On the other hand, all the listed works provide limited validation in real industrial contexts, as noted also by Femmer *et al.* [11]. In some cases, e.g., Gleich *et al.* [14], validation datasets are limited, while in other cases, e.g., Tjong and Berry [20], datasets are annotated for defects by one of the authors instead of domain experts. Large data-sets annotated by experts were considered by Falessi *et al.* [10]. However, their focus is solely on redundancy defects (i.e., equivalent requirements), detected by means of information retrieval techniques. The task of finding couples of equivalent requirements is radically different from the one we are dealing with in our study, in which multiple linguistic defects occurring in single requirements are considered. To our knowledge, the more general industrial work on defect detection is the one presented by Femmer *et al.* [11], who experimented their tool named *Smella* on several datasets belonging to three companies. Although domain experts were interviewed to assess the effectiveness of the tool, analysis of the results was performed by two researchers.

Compared to these studies, in the current work the validation of the approach is performed on a large set of industrial requirements annotated by domain experts. Another novelty is that defect detection NLP techniques are implemented *in-house* by a domain expert.

3 Overview

To experiment the feasibility of using defect detection NLP techniques, the company allocated one VE (VE1, 1st author) dedicated to the task, ISTI-CNR provided an Expert in defect detection through NLP (NLP-E, 2nd author), and the University of Florence provided a second VE (VE2, 3rd author), who worked at the company as VE, and then moved to the academia. NLP-E considered that assessing the effectiveness of a domain-generic tool for defect detection (e.g., QuARS [15]) would have required a strong *expertise* in the domain of the requirements documents. In addition, he considered that, if the tool would have provided too many false positive cases, e.g., *innocuous ambiguities* [7], the company would not have considered the tool as appropriate for its needs. Hence, it was decided to let VE1 develop the tool *in-house*, with the support of NLP-E. VE1 was initially required to study the papers of Berry *et al.* [4], Gnesi *et al.* [15], Gleich *et al.* [14], Tjong and Berry [20] and Arora *et al.* [2]. Then, she was required to perform the tutorials provided by GATE (General Architecture for Text Engineering [8]), which was the generic NLP tool selected to be tailored to support defect detection. The tool was chosen since it was considered sufficiently easy to use for an engineer, and sufficiently powerful for the task. After this autonomous training, VE1 and NLP-E met to define the defect classes on which to focus (Sect. 4). Priority was given to those defect classes that were considered more relevant from the point of view of VE1, and whose identification was considered feasible by NLP-E. For each defect class, VE1 used GATE to define a set of *patterns* for identification of defects. The patterns were experimented on a dataset annotated by VE1 herself, with the objective of maximizing recall, as suggested by Berry *et al.* [3] (Sect. 5). After the first encouraging results, a large-scale experiment was conducted on 1866 requirements, previously annotated by another VE of the company (VE3, 6th author) (Sect. 6). In this case, the results appeared particularly poor in terms of precision. Hence, VE1 and VE2 decided to analyse the false positive cases (Sect. 7). This analysis showed that many *true* linguistics defects were not considered in the validation performed by VE3. After marking these cases as *true positives*, several false positive cases remained, which could be in principle addressed by further tailoring the patterns to the specific language of the company. At the end of the experience, *all* the authors discussed about the lessons learned from the case study (Sect. 8).

4 A Rule-Based Approach to Predict Defects

4.1 NLP Technologies

Before describing the patterns that we defined to identify the defects, it is useful to list the natural language processing (NLP) technologies included in the tool GATE [8] that was adopted to define the patterns:

- **Tokenization:** This technology partitions a document into separate *tokens*, e.g., words, numbers, spaces, and punctuation.
- **Part-of-Speech (POS) Tagging:** This technology associates to each token a Part-of-Speech, e.g., noun (NN), verb (VB), adjective (JJ), *etc.* Common POS taggers are statistical in nature, i.e., they are trained to predict the POS of a token based on a manually annotated corpus.
- **Shallow Parsing:** This technology identifies noun phrases (NP) – in this case we speak about Noun Chunking – and verb phrases (VP) – in this case we speak about Verb Chunking – in sentences. For example, given the sentence *Messages are received by the system*, a shallow parser identifies {*Messages*, *the system*} as NP, and {*are received*} as VP.
- **Gazetteer:** This technology searches for occurrences of terms defined in a list of terms. In our case, we used it to check the presence of vague terms.
- **JAPE Rules:** This technology allows defining rules (i.e., high-level regular expressions) over tokens and other elements in a text [8]. A rule identifies sequences of elements that match the rule. Rules are expressed in the intuitive JAPE grammar, which is similar to regular expressions. JAPE rules can be rather long to report. In this paper, for the sake of space, to describe JAPE rules we will use a more concise and intuitive pseudo-code inspired to the JAPE grammar. In JAPE, and in our rules, the following symbols are used: "|" indicates logical or; "," indicates logical and; "!" indicates logical not; "<*expr*>+" indicates one or more elements matching the preceding expression *expr*; "<*expr*>∗" indicates zero or more elements; "<*expr*>?" indicates zero or one elements. When we use a term in capital letters, this indicates a form of *macro* that identifies terms of the specific type, e.g., NUMBER identifies

numbers, while ELSE identifies the term *else* in its various orthographic forms. Although these macros differ in terms of semantics, we expect that the reader can infer their meaning.

Table 1. Pattern adopted for each defect class.

Defect class	Pattern
Anaphoric ambiguity	P_{ANA} = (NP)(NP)+ (Split)[0,1] (Token.POS == PP | Token.POS =∼ PR*)
Coordination ambiguity	P_{CO_1} = ((Token)+ (Token.string == AND | OR)) [2] P_{CO_2} = (Token.POS == JJ) (Token.POS == NN | NNS) (Token.string == AND | OR) (Token.POS == NN | NNS)
Vague terms	P_{VAG} = (Token.string $\in$ Vague)
Modal adverbs	P_{ADV} = (Token.POS == RB | RBR), (Token.string =∼ "[.]*ly$")
Passive voice	P_{PV} = (AUXVERB)(NOT)?(Token.POS == RB | RBR)? (Token.POS ==VBN)
Excessive length	P_{LEN} = Sentence.len > 60
Missing condition	P_{MC} = (IF)(Token, !Token.kind == punctuation)* (Token.kind == punctuation)(!(ELSE | OTHERWISE))
Missing unit of measurement	P_{MU_1} = (NUMBER)((Token)[0, 1](NUMBER))?(!MEASUREMENT) P_{MU_2} = (NUMBER)((Token)[0, 1](NUMBER))?(!PERCENT)
Missing reference	P_{MR} = (Token.string == "Ref")(Token.string == ".") (SpaceToken)?(NUMBER)
Undefined term	P_{UT} = (Token.kind == word, Token.orth == mixedCaps)

4.2 Patterns for Defect Prediction

This section lists the classes of language defects considered, together with the patterns (i.e., JAPE rules) defined to identify them. Patterns are defined in terms of sequences of tokens to be matched within a requirement. Hence, the output produced by one pattern when applied to a requirement is zero or n requirement fragments (i.e., contiguous sequences of tokens in the requirement) that match the pattern. The patterns were defined by VE1 with the idea of identifying the defects that she perceived as more relevant for her job, and taking into account the defect classes provided by Berry *et al.* [4], and by the other papers she had studied [2,14,15,20]. In Table 1 we report the patterns in a compact version. The JAPE implementation of the patterns is available in our public repository[1]. Below, we describe the defect classes addressed by each pattern.

[1] https://github.com/BenedettaRosadini/QuARS-/tree/master/jape.

- **Anaphoric ambiguity.** Anaphora occurs in a text whenever a pronoun (e.g., *he*, *it*, *that*, *this*, *which*, etc.) refers to a previous part of the text. The referred part of the text is normally called *antecedent.* An anaphoric ambiguity occurs if the text offers more than one antecedent options [21], either in the same sentence (e.g., *The system shall send a message to the receiver, and* ***it*** *provides an acknowledge message* - *it* = *system* or *receiver*?) or in previous sentences. The potential antecedents for the pronouns are noun phrases (NP), which can be detected by means of a shallow parser. The pattern P_{ANA} matches any sequence of two or more noun phrases (NP), followed by zero or one sentence separators (Split), followed by a personal pronoun (PP), or other types of pronouns (PR*).
- **Coordination ambiguity.** Coordination ambiguity occurs when the use of coordinating conjunctions (e.g., *and* or *or*) leads to multiple potential interpretations of a sentence [7]. Two types of coordination ambiguity are considered here. The first type includes sentences in which more than one coordinating conjunction is used in the same sentence (e.g., *There is a* 90° *phase shift between sensor 1* ***and*** *sensor 2* ***and*** *sensor 3 shall have a* 45° *phase shift*). The second type includes sentences in which a coordinating conjunction is used with a modifier (e.g., *Structured approaches and platforms* – *Structured* can refer to *approaches* only, or also to *platforms*). The VE defined two patterns, one for each type. P_{CO_1} matches exactly two occurrences (notation "[2]") of one or more Tokens followed by a coordinating conjunction. P_{CO_2} matches cases in which an adjective (JJ) precedes a couple of singular (NN) or plural nouns (NNS), joined by *and* or *or.*
- **Vague terms.** Vagueness occurs whenever a sentence admits borderline cases, i.e., cases in which the truth value of the sentence cannot be decided [4]. Vagueness is associated with the usage of terms without a precise semantics, such as *minimal*, *as much as possible*, *later*, *taking into account*, *based on*, *appropriate*, etc. In our context, we use the list of 446 vague terms provided by the QuARS tool [15]. The list includes single-word and multi-word terms that were collected as source of vagueness in requirements. P_{VAG} matches any term included in the set *Vague* of vague terms.
- **Modal adverbs.** Modal adverbs (e.g., *positively*, *permanently*, *clearly*) are modifiers that express a quality associated to a predicate. As noted by Gleich *et al.* [14], adverbs are discouraged in requirements as potential source of ambiguity. VE1 noticed that, in the requirements of the company, most of the adverbs causing ambiguity were modal adverbs ending with the suffix -*ly*. For this reason, P_{ADV} matches adverbs in normal form (RB) or in comparative form (RBR) that terminate ($ indicates string termination) with -*ly*.
- **Passive voice.** The use of passive voice is a defect of clarity in requirements, and can lead to ambiguous interpretations in those cases in which the passive verb is not followed by the subject that performs the action expressed by the verb (e.g., *The system shall be shut down* – by which actor?). Passive voice detection is also considered by Gelich *et al.* [14] and by Femmer *et al.* [12]. To identify passive voice expressions, P_{PV} matches auxiliary verbs followed by a verb in past participle (VBN), possibly with negations and adverbs.

- **Excessive length.** Longer sentences are typically harder to process than short sentences, and can be source of unclarity. The VE decided to identify all the sentences that are longer than 60 tokens. Although this is a rather weak threshold – for generic English texts, Cutts recommends not to exceed 40 tokens [9] –, the VE considered this value appropriate for the length of the sentences in her domain.
- **Missing condition.** To be considered complete, each requirement expressing a condition through the *if* clause, shall have a corresponding *else* or *otherwise* clause. P_{MC} checks whether an *if* clause is followed by an *else/otherwise* clause in the same sentence.
- **Missing unit of measurement.** Each number is required to have an associated unit of measurement, unless the number represents a reference (see below). Hence, the patterns check whether a number has an associated unit, or a percentage value associated to it.
- **Missing reference.** This defect occurs when a reference that appears in the text in the form *Ref.* <X> does not appear in the list of references of the requirements document. To detect this defect we leverage the pattern P_{MR} to extract references in the text, and then – through Java code not reported here – we check whether each number found appears in the list of references.
- **Undefined term.** This pattern searches all the terms that follow the textual form used in the company for defining glossary terms (e.g., *restrictiveAspect*), which are expressed in camelCase format. As for the *missing reference* case, we leverage the P_{UT} pattern to search for terms expressed in camelCase (i.e., *mixedcap* orthography), and then we automatically search the glossary to check whether the term is present or not.

5 Preliminary Study

After the definition and implementation of the patterns, we performed a first assessment of the patterns on a real-world dataset of the company. In this phase, the goal was to establish whether the patterns were able to achieve a value of recall close to 100%. As noted by Berry *et al.* [3], defect detection techniques shall favor recall over precision since the cost of undetected *true* defects is much higher than the cost of manually discarding false positive cases. To perform the evaluation, the dataset was first manually annotated by VE1, and then she compared the output of the patterns with her annotations. In the following, we describe the annotation process, the evaluation measures adopted, and the observation on the results obtained.

5.1 Dataset and Annotation

For the analysis, a dataset of 241 system requirements was considered. This dataset was randomly selected from the requirements document of a wayside Automatic Train Protection (ATP) system and from the requirements document of an interlocking system. VE1 annotated the dataset. The requirement was labeled as *accepted* if it appeared to fulfill the criteria normally adopted by the company. These criteria are derived from the more general guidelines provided by the CENELEC EN 50128:2011 norm [6]. In particular a requirement was labeled as *accepted* if it was: (a) *feasible*: what is required is physically and technologically possible, can be done with available resources and is not against laws and regulations; (b) *testable*: can be demonstrated through repeatable tests or is at least verifiable through inspection; (c) *complete*: stand-alone, no missing references, undefined terms, to-be-defined parts, or missing conditions; (d) *clear and unambiguous*; (e) *uniquely identifiable*; (f) *consistent*: no internal contradiction and no contradiction with other requirements. The requirement was labeled as *rejected* in case it did not fulfill one of the criteria. In case the requirement was marked as *rejected* for criterion (c) or criterion (d), VE1 stated whether the rejection was due to one or more linguistic defect classes associated to the patterns listed in Sect. 4.2. In this case, VE1 labelled as $defective(i)$ each requirement fragment that included the i-th defect. After this annotation activity, 120 requirements were marked as *rejected*, while 121 were marked as *accepted*[2].

5.2 Evaluation Measures

Evaluation Measures by Defect. To measure the effectiveness of the patterns, we first provide a set of measures that focus on single defective fragments identified by the patterns. Given the pattern associated to the i-th defect, we consider the amount of true positive tp^D as the number of requirements fragments labeled as $defective(i)$ and correctly identified by the pattern; the amount of false positive fp^D as the number of requirements fragments wrongly identified as defective by the pattern; the amount of false negative fn^D as the number of requirements fragments labeled as $defective(i)$ that are not discovered by the pattern. Based on these definitions, we define the measure of precision (p^D) and recall (r^D) as:

$$p^D = \frac{tp^D}{tp^D + fp^D} \quad r^D = \frac{tp^D}{tp^D + fn^D}$$

The precision p^D is negatively influenced by the amount of defects wrongly identified (fp^D). The recall r^D is negatively influenced by the amount of undetected defects (fn^D).

Evaluation Measures by Requirement. To have a view of the effectiveness of the patterns applied together, we provide a set of measures that focus on the number of requirements, instead of on the number of defective fragments.

Here, we consider the amount of true positive tp^R as the number of requirements labeled as *rejected* for which at least one of the patterns correctly identified a defective requirement fragment; the amount of false positive fp^R as the number of requirements wrongly identified as defective (i.e., at least one of the

[2] The dataset appears balanced since VE1 continued to select requirements until a balanced number of accepted and rejected requirements was obtained.

patterns triggered a defect while the requirement was marked as *accepted*); the amount of false negative fn^R as the number of requirements marked as *rejected* for which none of the patterns triggered a defect. The measures of precision p^R and recall r^R are defined as for p^D and r^D, but considering tp^R, fp^R, and fn^R.

5.3 Results and Observations

In Table 2 we report the different evaluation measures. We see that, although the patterns for *anaphoric ambiguity* and *coordination ambiguity* are both based on shallow parsing, which normally has an accuracy of 90–95% [16], we achieve the objective of 100% recall. Similarly, for *modal adverbs* and *passive voice*, we achieve 100% recall, although these patterns employ POS tagging, which has an accuracy around 97% [18]. Two of the patterns that employ only lexical-based pattern matching, namely *missing reference* and *undefined term*, also achieve 100% recall. Lower values of recall are instead achieved for the patterns associated to *vague terms* (67.74%), *excessive length* (60.06%), *missing unit of measurement* (50%) and *missing condition* (97.05%).

Table 2. Preliminary study results for single defects and requirements.

Defect Class	tp^D	fp^D	fn^D	p^D	r^D
Anaphoric ambiguity	22	8	0	73.33%	100%
Coordination ambiguity	16	8	0	66.66%	100%
Vague terms	21	16	10	56.75%	67.74%
Modal adverbs	28	14	0	66.66%	100%
Passive voice	343	60	0	85.11%	100%
Excessive length	200	30	133	86.95%	60.06%
Missing condition	66	14	2	82.5%	97.05%
Missing unit of measurement	2	2	2	50%	50%
Missing reference	10	0	0	100%	100%
Undefined term	208	76	0	73.23%	100%
Requirements	tp^R	fp^R	fn^R	p^R	r^R
	106	59	14	64.24%	88.33%

- *Vague terms.* By inspecting the ten false negative defects for vague terms, VE1 found that they were all due to the absence of the quantifier *some* in the list of vague terms provided by QuARS. Hence, requirements such as the following were not marked as defective by the pattern: *In case the boolean logic evaluates the permissive state, the system shall activate* ***some*** *redundant output* – which output shall be activated? VE1 resolved the problem by simply adding the term *some* to the list of vague terms. Since also p^D was particularly low (56.75%), VE1 inspected the false positives and saw that they were due to domain-specific terms, namely ***raw*** *data*, ***hard*** *disk*, ***short-****circuit*, ***logical*** *or*, ***logical*** *and*, ***green*** *LED*. These terms were added to a stop-list to discard false positives in future analysis.

- *Excessive length.* By inspecting the false negative cases for excessive length, VE1 saw that they were due to a limitation of the GATE Tokenizer. For nested bullet point lists, the Tokenizer considers each item as a separate sentence. Hence, very long and deeply nested bullet point lists were not considered as sentences of excessive length. However, VE1 also argued that the length of a sentence, and the hard readability due to complex nested lists are different kinds of defects. Hence, she decided not to change the pattern for excessive length, and to consider the problem of nested lists as a defect that, at the moment, was left uncovered.
- *Missing unit of measurement.* Concerning the two false negative cases for missing unit of measurement, VE1 observed that these were due to the presence of ranges of numerical values, e.g., *[4,20]*, without the specification of the unit of measurement. To address these cases, the pattern was adjusted.
- *Missing condition.* The two false negative cases for missing condition appeared to be due to the presence of multiple *if* statements in the same sentence, with one *else* statement only, as in the following case: ***If** the initialization starts, **if** the board is plugged in and **if** the operator has sent the running command the system shall start, **else** it shall go in failure mode.* For requirements as the one presented, it is difficult to understand which specific *if* is covered by the *else* statement. Since the majority of missing condition defects were identified (66 out of 68), and considering that a VE has to manually review the requirements anyway, as required by the norm [6], VE1 decided not to add additional rules for this defect class.

False Negative Requirements. It is also useful to look at the values of false negative cases fn^R and recall r^R for the requirements. These 14 false negative cases not only include those already discussed, but also cases of defective requirements that could not be identified with our patterns – but which were annotated by VE1 following the guidelines of the company. In particular, interesting cases are those in which we have *inconsistent requirements* (e.g., 1: *The system shall accept only read access to file X*; 2: *The system shall accept read and write access to file X.*) that violate guideline (f), which asks requirements to be *consistent.* Other cases are those for which we have problems of *testability* (guideline (b)), as in the case of *under-specified statements* (e.g., *The system shall go in error mode when an internal asynchronism has been detected*; asynchronism among which components?), or *incomplete statements* (e.g., *The system shall make available its internal status*; through which interface?). Finally, other cases are those associated to other defects of completeness of the requirements document, as in the case of requirements for which it is expressed only the best-case scenario, and not the worst-case (e.g., *The system shall go at runtime state from power off state in 3 min in the best case.*; which is the requirement for the worst case?). Although some false negative cases were found, the evaluation of the patterns was considered successful in terms of recall by VE1. Hence, we decided to experiment the use of the patterns on a larger requirements dataset.

6 Large-Scale Study

The objective of the second study was to perform an assessment of the patterns on a larger requirements set of the company, previously validated by another VE (i.e., VE3), to understand to which extent the approach could be applicable more widely within the company.

6.1 Dataset, Annotations and Evaluation Measures

For this study a dataset of 1866 requirements was considered. The requirements belonged to a requirements document concerning a system that includes an interlocking, an ATP, a CTC (Centralised Traffic Control) and an Axle Counter. The defects of the document were previously annotated by VE3, following the criteria of the company already outlined in Sect. 5.1, and employed by VE1 for the preliminary study. Since this task was performed before this work was conceived, the annotation of the defective fragments was not performed by VE3, who just marked requirements as *accepted* or *rejected*, and described the reasons for rejection in a specific requirements validation document. From the 1866 requirements, 1733 were marked as *accepted*, while 93 were marked as *rejected*.

For the annotations performed by VE3, the measures adopted for evaluating the effectiveness of the patterns in identifying defective requirements are tp^R, fp^R, fn^R, p^R and r^R as defined in Sect. 5.2. Intuitively, these measures indicate whether the application of the different patterns simultaneously allows to identify requirements that were marked as *rejected* by VE3. Since VE3 did not annotate fragments, for this analysis we do not consider evaluation measures for the single defects as in the first analysis.

6.2 Results and Observations

In Table 4 we report the output of the patterns on the dataset in terms of defects identified (**D**), and in terms of defective requirements (**R**) – the other columns of the table will be discussed in Sect. 7. We see that the majority of the defects are due to *passive voice*. This is in line with the results of Femmer *et al.* [12]. The use of passive voice appears to be a sort of writing style of these requirements, since 615 out of 1866 (33%) include this defect. However, the most interesting – and disappointing – aspect comes from the evaluation presented in Table 3. The number of false positive requirements is extremely high, and the precision is only 5.7%. This value is comparable with the precision obtained through a random predictor [1] (for which $p^R = r^R = 93/1866\% = 5\%$). Hence, it appears not acceptable if the tool needs to be used in a real-world setting. Furthermore, also the value of r^R (74.19%) is not too encouraging. Hence, let us first focus on false negative cases, which impact the value of r^R, and in Sect. 7 we will discuss the analysis performed on false positive cases, which impact on p^R.

Table 3. Large-scale analysis results: requirements.

tp^R	fp^R	fn^R	p^R	r^R
69	1148	24	5.7%	74.19%

False Negative Cases. As for the preliminary analysis, the false negative cases are due to requirements that include defects that were not considered by any of the patterns, but that violate one or more criteria adopted by the company. Interesting examples are requirements that do not fulfill the criterion of *testability* (guideline (b)), as e.g., *The system shall be in continuous operation for 24 h a day and 7 days a week*; requirements that are not *feasible* (guideline (a)), e.g., *The core of the system shall use TCP/IP protocol in order to communicate with peripheral boards* – in this case, this requirement was considered not feasible since the only communication protocol that was considered applicable was UDP; requirements that include *inconsistent* statements (guideline (f)), e.g., *The brake symbol shall be able to show the following colors: Green when the brake is not active, Grey when the brake is not active.* Overall, these cases show that there is a variety of defects that are hardly identifiable with NLP techniques, and hence require a human expert to accurately assess them.

7 False Positive Analysis

Given the poor results in terms of precision, VE1 inspected the output of the tool, and saw that part of the false positive requirements were, in her opinion, actually defective. For example, the following requirement marked as *accepted*, was evidently defective due to several vague terms (highlighted in bold): ***Depending on*** *the technical or functional solution selected, there shall be time parameters in the control system, that the Purchaser shall be able to adjust during operation in order for the registration/deregistration to be made* ***as effectively as possible.***[3] In other terms, her opinion was that VE3, when evaluating the requirements, actually tolerated several linguistic defects, and marked as *rejected* only those requirements that appeared to include severe conceptual defects. To assess how many of the false positive cases could be considered as linguistic defects from the point of view of a more strict annotator, a second annotation process was performed to evaluate the false positive cases.

7.1 Annotation and Evaluation Measures

A second annotation process was performed on the requirements marked as defective by at least one of the patterns. In this annotation process, two VEs (VE1 and VE2) independently annotated the output of the patterns as follows. For each requirement fragment labelled as defective according to pattern i, each

[3] The requirement was not rejected since it was clarified by other subsequent requirements. This violates the guideline (c) that require requirements to be stand-alone, but the defect was not considered crucial.

VE annotated the fragment as $defective(i)$, if the VE considered the defect as a true defect. The annotator agreement was estimated with the Cohen's Kappa [17], resulting in $k = 0.8225$, indicating an almost perfect agreement. Overall, if a fragment was annotated as $defective(i)$ by at least one annotator, the fragment was marked as $defective(i)$ in the annotated set used for the evaluation. In this analysis, we use evaluation measures for single defects, and for entire requirements. Since in this analysis we focus solely on the output produced by the patterns, we consider neither the amount of false negative cases, nor the measure of recall (for this reason the structure of Table 4 differs from that of Table 2). Hence, we consider p^D (for each defect class i) and p^R as defined as in Sect. 5.2.

Table 4. Evaluation of the results for the large-scale study.

Defect class	**D**	**R**	tp^D	fp^D	p^D
Anaphoric ambiguity	387	327	258	129	66.6%
Coordination ambiguity	263	213	190	73	72.24%
Vague terms	496	306	290	206	58.46%
Modal adverbs	476	373	331	145	69.53%
Passive voice	1265	615	1242	23	98.1%
Excessive length	16	16	16	0	100%
Missing condition	188	148	129	59	68.61%
Missing unit of measurement	0	0	0	0	-
Missing reference	4	2	4	0	100%
Undefined term	54	49	43	11	79.62%
Average					**79.24%**
Requirements			tp^R	fp^R	p^R
			1042	175	**85.6%**

7.2 Results and Observations

Table 4 reports the results of this phase. For each defect class, the precision reaches an average value of **79.24%** for what concerns the number of defects (average of different p^D). Overall p^R resulting from the application of all the patterns together, raises from the 5.7% of Table 3, to **85.6%**. However, there is still a significant amount of false positive cases that should be noticed. For the sake of space, we will present examples for *vague terms*, since these are the defects for which the false positive cases had a major impact on the precision value ($p^D = 58.46\%$). False positive cases of anaphoric ambiguity are studied by Yang *et al.* [21], while Chantree *et al.* [7] studied false positive cases of coordination ambiguity. Our false positive cases for these defect classes are similar to those addressed by these studies. For modal adverbs, false positives occur when adverbs form domain-specific names, e.g., ***normally*** *closed* to refer to relay status.

Vague Terms. A large number of false positive cases (206) is identified for this defect. These cases are due to the fact that many of the vague terms are lexically ambiguous. For example, the term *light*, considered as adjective, is vague, but when playing the role of noun, as in the requirement *Yellow Stop **lights** do not have to be monitored*, is not vague. Cases such as the one in this example can be potentially detected by applying POS tagging, and considering a term as vague only if it plays the role of adjective. Other cases occur when a vague word is part of a domain-specific multi-word term, as for the term *distant* of the following example: *The operator shall use "**distant** signalling distance" to apply the brake.* To discard these cases, techniques for multi-word term identification [5] should be applied. Finally, many cases were due to the usage of the term *possible* in the phrase *It shall be **possible** [...]*, considered an accepted requirement preamble within the company. This phrase was included in a stop-phrase list, to discard false positives, and allowed to increase the precision p^D for vague terms from 58.46% to **78.37%** (about 20% increase). This shows that small adjustments to the patterns can radically improve the results in terms of precision, since requirements appear to present *systematic* sources of false positives.

8 Discussion and Conclusion

This paper presents the experience of a railway signalling manufacturer in implementing a set of NLP patterns to detect defects in NL requirements. From the experience, a set of lessons learned were discussed among the authors, and are reported below.

In-house NLP. Our experience shows that NLP technologies are available for requirements analysts with limited NLP training, and that these technologies can be proficiently used for the detection of several typical requirements defects. Rule-based NLP patterns tend to generate large numbers of false positives [7,21]. If the results come from a tool that the requirements analyst cannot control, the analyst is likely to distrust the tool. Instead, if the analyst understands the inherent principles of the tool – and implementing the tool is a proper way for understanding its principles –, s/he can understand its weaknesses and use it at its best. Furthermore, it is also important to internally develop the tools, since, to reduce the amount of false positive cases, tailoring the patterns for the specific needs of the company is required. If the VE implements the patterns, s/he can customise them according to the language used in the domain, as, e.g., to account for terms such as *raw data*, *hard disk* (Sect. 5.3), and phrases such as *it shall be possible*. This last customisation allowed to increase p^D for vague terms by 20% (Sect. 7.2).

Requirements Language Counts. Looking at the large number of passive voice defects in the large-scale analysis, it appeared that the use of passive voice was a form of writing style. As a consequence, the patterns generated a large number of detected defects (i.e., 1265). This tells us that, to effectively use NLP, one cannot simply implement appropriate defect detection patterns: one should change also the language adopted in the requirements, to make it more error free, so that the VE can focus on a smaller amount of defects. For this reason, we argue that NLP tools should be first used by the requirements editors, to limit the amount of poor writing style, and only *afterwards* by a VE. However, this is not always practicable, especially in those cases in which requirements are produced by the customer, and assessed by the company who has to develop the product.

Validation Criteria Count. Comparing the results of the preliminary analysis with those of the large-scale study, we saw that a large part of the false positive cases encountered in the second analysis could be associated with a weaker validation performed by VE3, who did not focus on linguistic defects, but more on severe conceptual defects. For this reason, the results obtained in terms of precision were extremely poor. When changing criteria (Sect. 7), p^R varied from 5.7% to 85.6%. Hence, to perform an appropriate validation of rule-based NLP patterns, it is advisable to start from an annotated dataset that has been defined *knowing* the classes of defects that will be checked by the patterns. Otherwise, the results might be misleading. This observation might appear counter-intuitive, since we suggest to adapt human operators to tools. However, when dealing with the complexity of NL, we argue that the adaptation between humans and NLP tools should be bi-directional.

NLP is Only a Part of the Answer. In our large-scale study, several false negative cases occurred, which can hardly be detected with NLP. These are examples of conceptual defects that require a human with knowledge of the domain and of the specific project. The amount of these cases – 24 out of 93 defects in total – is not negligible. Furthermore, it is worth noting that 69 out of 93 conceptual defects could be actually detected by looking at *linguistic* defects that can be identified with NLP. Although computing the correlation between linguistic defects and conceptual defects is out of the scope of this work, this result suggests that some form of relation between the two might exist, and this is an aspect that is worth further exploration.

Statistical NLP vs Lexical Techniques. Our patterns make use of POS tagging and shallow parsing, which are statistical techniques that can hamper the objective of 100% recall [3]. However, in Sect. 5, we showed that 100% recall was achieved for those patterns that used these techniques, while it was *not* achieved for the pattern adopted for *vague terms*, which uses a lexical based approach. Hence, we argue that the argument in favour of a"dumb" lexical-based defect detection approach instead of an approach that leverages statistics-based technique [3] should be partially revised. If one wants to use lexical-based detection approaches, then one should use only defect indicators belonging to closed word classes (e.g., pronouns, conjunctions). Instead, if one uses open word classes (e.g., adjective, adverbs), the problems are not different from those that *might* emerge with statistical techniques. As these latter may fail, also lists of dangerous adjectives and adverbs may fail, because they might not include words that were not considered until they appear in the requirements (as e.g., the word *some*, as noted in Sect. 5.3).

Overall, the experience was considered extremely useful by the company. In particular, VE1 says that, after studying the literature on defect identification, and implementing the patterns, also her way of judging requirements defects became more strict. This is also the reason why requirements marked as *accepted* by VE3, were afterwards *rejected* by VE1 and VE2. In future works, appropriate adjustments will be defined to address the false positives identified in this study. Concerning false negative cases, it is worth remarking that, unless the tool for defect detection is appropriately validated, a VE has to manually inspect the requirements anyway to produce the verification report, as required by the CENELEC EN 50128:2011 norm [6]. Although human review cannot be replaced, NLP support can help a VE in *prioritising* the requirements to be manually analysed for defects, or, as suggested by Berry *et al.* [3], to check for defects left behind after a manual analysis has been performed.

References

1. Alvarez, S.A.: An exact analytical relation among recall, precision, and classification accuracy in information retrieval. Technical report BCCS-02-01. Computer Science Department, Boston College (2002)
2. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Automated checking of conformance to requirements templates using natural language processing. IEEE TSE **41**(10), 944–968 (2015)
3. Berry, D., Gacitua, R., Sawyer, P., Tjong, S.F.: The case for dumb requirements engineering tools. In: Regnell, B., Damian, D. (eds.) REFSQ 2012. LNCS, vol. 7195, pp. 211–217. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28714-5_18
4. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: linguistic sources of ambiguity (2003)
5. Bonin, F., Dell'Orletta, F., Montemagni, S., Venturi, G.: A contrastive approach to multi-word extraction from domain-specific corpora. In: LREC 2010 (2010)
6. CENELEC: EN 50128: 2011: Railway applications - communication, signalling and processing systems - software for railway control and protection systems. Technical report (2011)
7. Chantree, F., Nuseibeh, B., Roeck, A.N.D., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: RE 2006, pp. 56–65 (2006)
8. Cunningham, H.: GATE, a general architecture for text engineering. Comput. Humanit. **36**(2), 223–254 (2002)
9. Cutts, M.: The Plain English Guide. Oxford University Press, Oxford (1996)
10. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Trans. Softw. Eng. **39**(1), 18–44 (2013)
11. Femmer, H., Fernández, D.M., Wagner, S., Eder, S.: Rapid quality assurance with requirements smells. J. Syst. Softw. **123**, 190–213 (2017)
12. Femmer, H., Kučera, J., Vetrò, A.: On the impact of passive voice requirements on domain modelling. In: ESEM 2014, p. 21. ACM (2014)
13. Ferrari, A., Gnesi, S.: Using collective intelligence to detect pragmatic ambiguities. In: RE 2012, pp. 191–200 (2012)
14. Gleich, B., Creighton, O., Kof, L.: Ambiguity detection: towards a tool explaining ambiguity sources. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 218–232. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14192-8_20
15. Gnesi, S., Lami, G., Trentanni, G.: An automatic tool for the analysis of natural language requirements. Comput. Syst. Sci. Eng. **20**(1), 53–62 (2005)

16. Kang, N., van Mulligen, E.M., Kors, J.A.: Comparing and combining chunkers of biomedical text. J. Biomed. Inform. **44**(2), 354–360 (2011)
17. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics **33**(1), 159–174 (1977)
18. Manning, C.D.: Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In: Gelbukh, A.F. (ed.) CICLing 2011. LNCS, vol. 6608, pp. 171–189. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19400-9_14
19. Mich, L., Franch, M., Inverardi, P.N.: Market research for requirements analysis using linguistic tools. REJ **9**(1), 40–56 (2004)
20. Tjong, S.F., Berry, D.M.: The design of SREE — a prototype potential ambiguity finder for requirements specifications and lessons learned. In: Doerr, J., Opdahl, A.L. (eds.) REFSQ 2013. LNCS, vol. 7830, pp. 80–95. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37422-7_6
21. Yang, H., Roeck, A.N.D., Gervasi, V., Willis, A., Nuseibeh, B.: Analysing anaphoric ambiguity in natural language requirements. REJ **16**(3), 163–189 (2011)