DeNom: A Tool to Find Problematic Nominalizations using NLP

Mathias Landhäußer,
Sven J. Körner,
Walter F. Tichy, and
Jan Keim
Karlsruhe Institute of Technology, Germany
landhaeusser, sven.koerner, tichy@kit.edu, jan.keim@student.kit.edu

Jennifer Krisch
Daimler AG, Germany
jennifer.krisch@daimler.com

***Abstract*—Nominalizations in natural language requirements specifications can lead to imprecision. For example, in the phrase "transportation of pallets" it is unclear who transports the pallets from where to where and how. Guidelines for requirements specifications therefore recommend avoiding nominalizations. However, not all nominalizations are problematic. We present an industrial-strength text analysis tool called DeNom, which detects problematic nominalizations and reports them to the user for reformulation.**

DeNom uses Stanford's parser and the Cyc ontology. It classifies nominalizations as problematic or acceptable by first detecting all nominalizations in the specification and then subtracting those which are sufficiently specified within the sentence through word references, attributes, nominal phrase constructions, etc. All remaining nominalizations are incompletely specified, and are therefore prone to conceal complex processes. These nominalizations are deemed problematic.

A thorough evaluation used 10 real-world requirements specifications from Daimler AG consisting of 60,000 words. DeNom identified over 1,100 nominalizations and classified 129 of them as problematic. Only 45 of which were false positives, resulting in a precision of 66%. Recall was 88%. In contrast, a naive nominalization detector would overload the user with 1,100 warnings, a thousand of which would be false positives.

I. Introduction

Engineers are no poets. Furthermore, they are under pressure to get to the "real work" of developing products. Hence, requirements specifications are often of mediocre quality. Because of that there is a need for tools that identify flaws in requirements specifications. Flaws in requirements specifications fall into three categories: deletion, generalization, and distortion [1]. Nominalizations belong to the category of distortion and are often overlooked. A nominalization is the noun form of a verb – which is not problematic by itself but it may make a complex process sound like a simple event. Authors tend to forget to mention necessary details when they use nominalizations. In "transportation of pallets", for example, "transportation" is the nominalized form of "to transport". But that hides from the reader that "transport" is a (potentially complex) process. Using the verb form "to transport" instead, the author is urged to answer the questions "Who?", "How?", and in that case also "From where to where?".

To minimize the number of defects in requirements specifications, many demand to use structured documents, templates, and to strictly follow writing rules. A common rule is not to use nominalizations. The rule "Thou shall not use nominalizations!" is fine when writing requirements. But the rule "Find and eliminate all nominalizations!" is too strict for inspections: One can give all the details and still use a nominalization. Identifying (and rewriting) acceptable nominalizations is tedious and unnecessary.

Previous work showed that linguistic flaws can be automatically detected [2]. Körner and Brumm presented RESI, a tool that scans documents for linguistic flaws and reports them to the user (see Section II-C). But RESI overloads the user with the sheer number of (possible) flaws. It reports all nominalizations including the acceptable ones. The false positive rate is high. Recent work [3] showed that RESI can be used to detect defects in real world requirements specifications, but the high number of false positives prohibits the actual use of the tool in a real-world scenario.

This paper presents four categories for nominalizations; only one of which contains truly problematic nominalizations (see Figure 1). We identified the categories by manually inspecting requirements from Daimler AG and found that most of the nominalizations classify as acceptable. We show in the second part of the paper how to identify the problematic nominalizations and how we built this functionality into DeNom. DeNom only reports problematic nominalizations to the user. Section II reviews related work. Sections III and IV detail the categories and the automatic classification. We are following the recommendations of Ivarsson with regard to improving relevance and rigor [29]. Section V shows an evaluation of DeNom with requirements from Daimler AG and the last section concludes the paper with future work.

II. Related Work

Many publications discuss problems in requirements specifications. They describe how to avoid and detect flaws and how to assess the quality of requirements specifications. We give a brief overview of these three areas.

978-1-5090-0125-5/15

AIRE 2015, Ottawa, ON, Canada

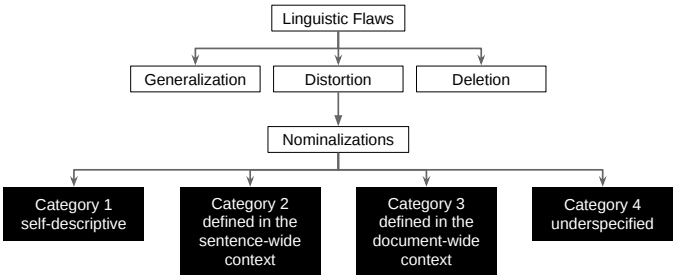


Fig. 1. A categorization of linguistic flaws. We split nominalizations in four categories.

TABLE I
RESULTS OF THE MANUAL PREPARATORY STUDY

	Words	Nom.	Cat. 1	Cat. 2	Cat. 3	Cat. 4
SRS 1	9,942	85	0.0%	70.6%	29.4%	0.0%
SRS 2	23,104	158	0.0%	59.5%	36.7%	2.5%
SRS 3	2,129	21	0.0%	81.0%	14.3%	0.0%
SRS 4	3,687	62	0.0%	95.2%	4.8%	0.0%
SRS 5	1,598	30	0.0%	56.7%	43.3%	0.0%
Sum	40,460	356	0	247	102	4
			0.0%	69.4%	28.7%	1.1%

A. Avoiding Flaws when Writing Requirements

Numerous approaches and advice exist on how to write good requirements. A well-known approach is to control the language. Controlled languages restrict grammar and/or vocabulary of the natural language to avoid problems. A widespread representative of controlled languages is Attempto Controlled English (ACE) [4]. Similarly, Denger et al. advocate pattern-based writing [5], but admit that converting existing requirements specifications to a pattern-based form can be difficult. Smith et al. combine formal models of the requirements (i.e. finite state machines) and requirements in a so-called disciplined natural language [6]. Therefore all requirements must be accompanied by state machines. Keeping them and the natural language requirements synchronized is a huge effort.

Berry et al. wrote a handbook on avoiding ambiguities, but their rules can also be used for inspections [7]. Rupp dedicates a chapter on linguistic problems in requirements [1]. She defines three categories of linguistic defects: Deletion, generalization, and distortion. Nominalizations are a distortion. Her advice is to adhere to writing rules and to use templates. A similar approach is EARS [8]. It provides five templates for different types of requirements and guides authors through the writing process. Pisan takes a different direction: He matches incomplete requirements to existing known-to-be-good requirements [9]. The good requirements are used to augment the requirements specification at hand. Yet, his method requires a database of good requirements.

B. Assessing the Quality of Requirements

Davis et al. evaluate 24 criteria with metrics to determine the overall quality of a requirements specification [10]. Some of the criteria affect or even contradict each other. Therefore the authors point out that a perfect requirements specification does not exist. Wilson et al. count the occurrences of certain expressions in a document to evaluate its quality [11]. Their quality indicators include completeness and consistency.

Fabbrini et al. present the Quality Analyzer for Requirement Specifications (QuARS), a tool that checks requirements specification using predefined word lists [12]. The lists give indicators for problems. QuARS marks a phrase as problematic if the number of indicators in the phrase exceeds a given threshold. Fantechi et al. use QuARS as conceptual base of their metrics-based analyses [13]. Berry et al. extended the quality model of QuARS [14].

C. Detecting Flaws in Requirements

There are some approaches to detect specific flaws in requirements specifications. Fagan pioneered the idea of design and code inspections [15]. Requirements inspections have been used for decades, but still hugely depend on the inspector's proficiency [16], [17], [18]. In 2000, Kamsties and Paech proposed an improved inspection type to detect ambiguities in natural language requirements [19]. Chantree et al. presented a tool that identifies ambiguities [20]. They train a classifier with word distributions and human judgments about whether the respective requirement is ambiguous or not. The classifier is used to report only problematic ambiguities for rewriting.

In 2008, Verma et al. presented their Requirements Analysis Tool (RAT) [21]. RAT is a word processor plug-in that analyzes natural language requirements by using a user-defined glossary and constrained language. RAT highlights problematic requirements directly in the requirements specifications, but requires training. Accenture had used RAT in larger project with some success, but the overall need of specifically trained users and the time consumption lead to the decommissioning.

Körner and Brumm present the Requirements Engineer's Specification Improver (RESI), a tool that checks requirements specifications for linguistic flaws [2]. RESI uses natural language processing (NLP) tools from Stanford, WordNet [22], and the ResearchCyc ontology [23]. It identifies flaws such as incomplete process words (i.e. a verb that lacks arguments), ambiguities (i.e. a word that has multiple meanings), and nominalizations. Through RESI's user interface, the user directly fixes flaws in the specification. Extensive case studies showed that RESI has an excellent recall and user studies showed that no special training is needed to use RESI [24].

In recent work, Krisch and Houdek discuss the impact of passive voice and weak words on requirements [25]. Both should be avoided when writing down requirements, but a simple text search is not suitable for an automatic flaw detector: whether passive voice and weak words are acceptable or not depends on the context in which they are used in. Passive voice is almost never problematic and only 12 % of the weak words lead to problematic requirements.

III. NOMINALIZATIONS: PROBLEMATIC OR NOT?

A nominalization becomes problematic when misinterpreted. This is mostly because it leaves out descriptive infor-

TABLE II
RESI'S RESULTS IN THE PREPARATORY STUDY

	Words	Nom.	Cat. 1	Cat. 2	Cat. 3	Cat. 4
SRS 3	2,158	25	0.0%	88.0%	8.0%	0.0%
SRS 4	3,687	56	0.0%	83.9%	1.8%	0.0%
SRS 5	1,598	15	0.0%	73.3%	6.7%	6.7%
SRS 6	6,069	116	0.0%	91.4%	5.2%	0.0%
SRS 7	12,581	133	0.0%	84.2%	5.3%	0.0%
SRS 8	7,403	154	0.0%	74.7%	16.2%	0.0%
Sum	33,496	499	0	413	42	1
			0.0%	82.8%	8.4%	0.2%

mation. Studies show that misinterpretation leads to problems in the development stage with the same impact as ambiguous wording, incompletely specified process words, and wrong quantifiers [26].

In a preparatory study, we inspected requirements specifications manually, then supported by RESI. The analysis showed that nominalizations can be classified in four different categories (see Figure 1). Nominalizations of categories one and two are sufficiently specified and therefore acceptable. The third category borders on being problematic depending on how explicitly context and domain can be determined from the corresponding requirement to sufficiently specify the possibly problematic nominalization. Category 4 is problematic since it does not include enough information. We manually determined the distribution of nominalizations across the 4 categories in eight specifications with a total of over 66.000 words. The requirements specifications come from various projects of Daimler AG. Tables I and II show the results of this preparatory study. Category 1 and category 4 nominalizations turn out to be rare.

Then, we implemented DeNom to improve RESI, which detects all nominalizations without categorizing them. The main focus in this paper lies on correctly categorizing category 2 and category 3 nominalizations to withhold these from the user: we expect a much better user acceptance of the tool once it shows problematic nominalizations only.

A. Category 1 Nominalizations: Acceptable

Category 1 nominalizations are self-descriptive and therefore unproblematic. They are fully specified independent of the sentence and do not need further references to other objects or sentences. Therefore, they cannot be misinterpreted. An example is the word "troubleshooting" where the sentence's subject referenced by the nominalization is easily inferred from context. Mostly, category 1 also includes words that are self-descriptive due to the specific domain they are used in. Words that are defined in a glossary are also in category 1.

B. Category 2 Nominalizations: Acceptable

Category 2 nominalizations can be determined through context, i.e. the including sentence carries words or phrases that refer directly to the nominalization and specify it. The category can be split in two sub-categories: One refers to the adjacent words and phrases also called a nominal phrase. The other refers to remote words and phrases within the same sentence.

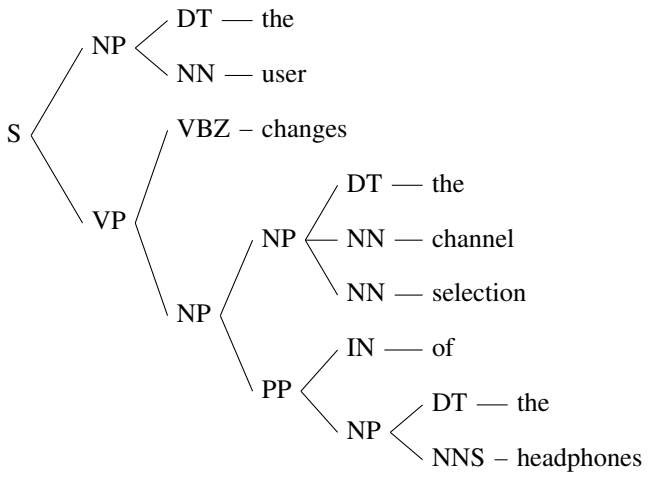


Fig. 2. Syntax tree for "[...] the user changes the channel selection [...]".

Nominal phrases can be a combination of nouns forming a compound noun. Compound nouns not only specify a nominalization in more detail, they often make them disappear since the nominalization now serves as descriptive attribute of another noun. An example is the sentence "[...] the user changes the channel selection of the headphones [...]" where the nominal phrase "channel selection (of the headphones)" specifies the nominalization "selection". "Selection" as a noun is an unspecified nominalization, but as compound noun "channel selection", "selection" is explained. Additionally, the prepositional phrase beginning with "of" further serves as a description by pointing the channel selection process to the headphones, thereby rendering it into a clear statement.

Figure 2 shows the syntax tree of the example sentence. A syntax tree shows the structure of the sentence; the leafs of the tree are the words of the sentence, the inner nodes express the composition of the sentence. In Figure 2 you can see that "the channel selection" forms a nominal phrase (NP), as indicated by their common parent node NP. The additional information "of the headphones" is added as prepositional phrase (PP). The noun phrase and the prepositional phrase form a new nominal phrase, which is the object of the verb "changes".

In many cases however, the reference of the nominalization is neither adjacent nor part of a nominal phrase. Yet these referenced words can also specify nominalizations further. An example would be: "For a given model series, the implementation of this function shall be discussed during development." The nominalization "development" could (depending on context) refer to the "model series" which would then specify the nominalization sufficiently.

C. Category 3 Nominalizations: Acceptable

Category 3 nominalizations are specified through domain or document context. This distinguishes them from possibly problematic statements. Occasionally, the context or the references of the nominalization are ambiguous. In the latter case, the respective nominalizations are problematic and are

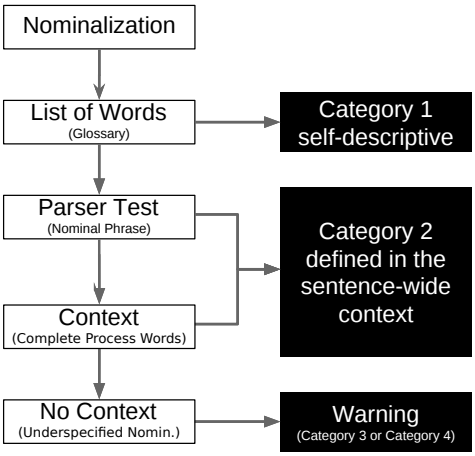


Fig. 3. A decision tree detailing the choices of DeNom.

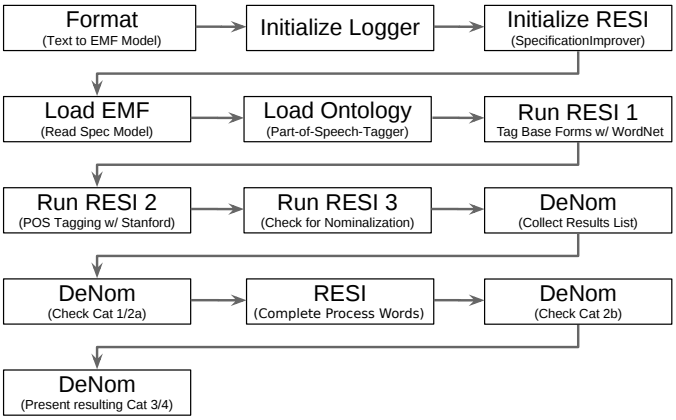


Fig. 4. Overview of DeNom's processing steps.

in category 4. This is why category 3 nominalizations border on being problematic. How problematic they really are can only be determined on a case by case basis. An example is "The exact design will be determined during development". If "development" refers to the product to be built, this might be clear for the human reader knowing the context. For a machine or a reader unfamiliar with the project, this is yet undefined. If uncertain, we treat category 3 nominalizations as problematic to stay on the safe side. With this, we expect a higher number of false positives. We show in the evaluation that the number of false positives appears reasonable.

D. Category 4 Nominalizations: Problematic

Category 4 nominalizations are problematic. They have to be changed in the specification. Their context (which shall describe them sufficiently) is ambiguous or non-existent. The missing information can lead to misinterpretations without the reader even noticing the mistake. The only solution: These nominalizations need additional information, or they need to be rephrased or removed.

An example is "For fine optimizations and verification an additional cycle in the B-phase car will be permitted." The context of the two nominalizations "optimization" and "verification" is not clear. It is unclear to which optimizations and verifications they refer to, even with the context. Since automatic software-based detection could never reach as far as a reader's knowledge about the process, DeNom tags these nominalizations as problematic.

IV. Building DeNom: Extending RESI to Categorize Nominalizations

Previous work showed that RESI reliably detects nominalizations (see Section II-C). The achieved recall is 100% in all published case studies, but RESI does not distinguish between *acceptable* and *problematic* nominalizations. DeNom uses RESI to detect nominalizations but further classifies them. To increase user efficiency and usability, DeNom processes RESI's results, but skips user interaction for categories 1 and 2, asks the user to specify category 3 nominalizations, and reports category 4 nominalizations as being problematic. Figure 3 depicts the work flow DeNom uses.

RESI works with specifications in a special format[1] that can hold text and additional information such as POS tags, links to ontologies, and markers for flaws. Figure 4 shows an overview of the processing steps. First, we export the specifications from IBM DOORS into text files. Exporting the specifications adds some extra textual information, which has to be formatted, arranged, or deleted. Therefore we push the extracted specifications through the DeNom formatter. It removes unnecessary characters and symbols, normalizes the use of apostrophes and other punctuation marks and makes the input stream digestible for RESI's text to EMF converter. Rather than using RESI's interactive user interface, DeNom configures and starts RESI in the background and does all the setup automatically. DeNom connects to the Cyc ontology, preprocesses the document with POS and base form taggers and then starts the inspection process (for details cf. [2]).

A. Category 1

Category 1 nominalizations can be detected easily by referencing the found nominalizations with a glossary. If DeNom does not detect category 1, it checks for category 2.

B. Category 2

Category 2 nominalizations are detected by examining if the nominalization is part of a nominal phrase. For implementation purposes, we do not use the syntax tree described in the concept, but the dependency graph of the corresponding sentence. According to Stanford's NLP group, dependency graphs are the preferable vehicle when processing word connections in a sentence, i.e. dependencies between these words. A dependency graph contains the words of a sentence as nodes and typed dependencies as edges. A typed dependency is a predicate `reln(gov, dep)` where `reln` is the relation (i.e. the type of the dependency), `gov` the governor of the relation, and `dep` the dependent: For example, `dobj(return, pallet)` means that "pallet" is the direct object of "return".

[1] It uses the Eclipse Modeling Framework (EMF) file format.

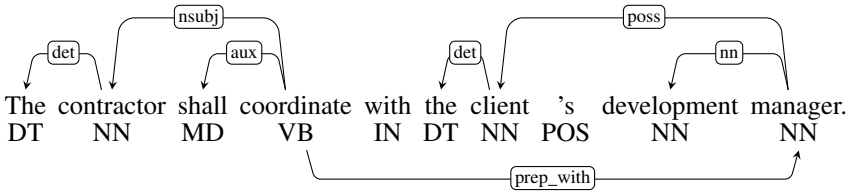


Fig. 5. The dependency graph for the first example (category two): The nominalization "development" forms a compound noun with "manager". The POS tags are given below the words.

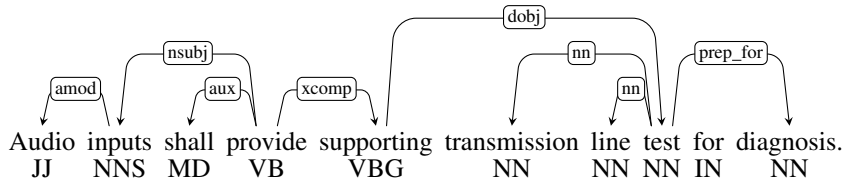


Fig. 6. The dependency graph for the second example (category two): There is a prep_for relation between the parent node and the nominalization. The POS tags are given below the words.

The directed edges span from the governor to the dependent, e.g. from "return" to "pallet". Dependency graphs provide a more direct access to the information in a sentence, whereas syntax trees provide access to the structure and the constituents of a sentence. There are 56 different typed dependencies and the graphs can be built with the Stanford parser [27].

Our first test checks if the parent node of the nominalization has been marked as noun (NN), which means it builds a nominal phrase with the nominalization. Also, the parent test needs to confirm that the grammatical relationship between the nominalization and its parents node are not of the type `prep_for` or `prep_of` since this entails that the nominal phrase is not sufficiently specifying the nominalization. E.g., in Figure 5 the nominalization is "development" but it forms a compound noun with "manager". If we detect a compound noun in the dependency graph, the nominalization is qualified to be in category 2.

A different case is the example in Figure 6: The graph for "Audio inputs shall provide supporting transmission line test for diagnosis" shows that the nominalization "diagnosis" is connected to "test" with the preposition "for". The word "test" itself does not specify the nominalization, rather does the nominalization specify the test further. The prepositions "for" and "of" are evidence that this is the case.

The existence of prepositions checking the child node relationship is similar. DeNom checks if at least one child node of the nominalization carries a noun identifier (NN). Figure 7(a) shows and example where the child nodes "channel" and the and "headphones" form a nominal phrase with the nominalization "selection". This specifies the nominalization sufficiently.

But nominalizations are not only specified through the noun to which they refer to. Therefore DeNom also checks if the grammatical relation `prep_by` exists. Figure 7(b) shows an example. The nominalization "transport" lacks the object, i.e. the information what exactly needs to be transported.

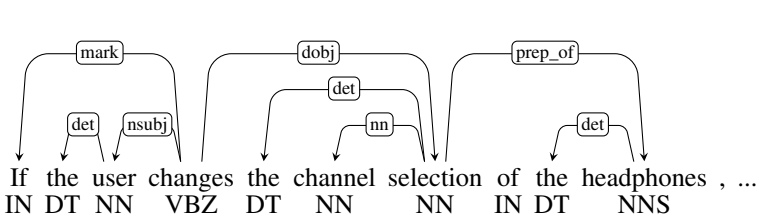
C. Category 3

Category 3 nominalizations are more complicated as we have to identify the needed information in the sentence. RESI can check whether process words (i.e. verbs) are fully specified, i.e. if all the arguments that the verb needs are given in the sentence. For instance, if the sentence states that "she returns the pallets", RESI asks "to whom?". The information about which verb arguments are necessary is found in the Cyc ontology. DeNom uses the same analysis to identify whether all needed information is given to sufficiently specify a nominalization. In order to do so, DeNom transforms the nominalization into the corresponding verb and then runs the analysis. In short this means: if we can find all necessary arguments for the respective process word, we can assume that the nominalization is fully specified and therefore acceptable.

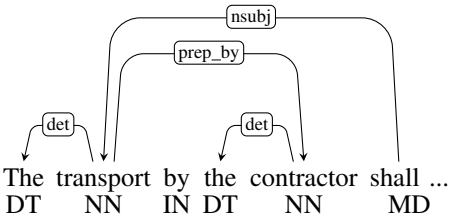
We repeat an example from reference [2] to give a precise idea of how DeNom and RESI work. We slightly adapted the following example for this paper; a detailed description of RESI's other rules can be found in reference [2]. The example sentence is:

```
Every pallet is returned after transport.
```

In the current setting, DeNom queries ResearchCyc using Cycs very own query language called CycL. Predicates and fixed terms are prefixed with `#$` and variables (which contain the response to the query) are denoted using a `?`. ResearchCyc is structured into so called Microtheories, which help to distinguish and to represent contradictory knowledge of common world objects as we encounter on a daily basis. An example would be the word "bank" in the context of a building for money business or a wooden structure to sit on. Also, it could just refer to the river bank. Each of these microtheories is tailored to encapsulate certain knowledge to allow for consistent reasoning and deduction within this Microtheory. DeNom queries the microtheory *GeneralEnglishMt*, which contains knowledge about the English language.



(a) Example category two nominalization (*selection*), which builds a nominal phrase with *channel* and *headphones*.



(b) A nominal phrase with a prep_by-relation between the nominalization and the child node does not qualify for category two. DeNom proceeds with the category three checks.

Fig. 7. Dependency graph examples.

We identify the argument lists for the nominalization (through its deducted verb) with a single query. But first, the process word (in this case `return`) has to be transformed into a word constant. This is done by capitalizing the first letter of the word and appending `-TheWord` to the corresponding word string. Then we ask ResearchCyc about the argument list:

```
(#$verbSemTrans #$Return-TheWord ?SENSECOUNTER
   ?FRAMETYPE ?FRAME)
```

The variable `?FRAME` then holds the argument list in the following format:

```
(#$and
   (#$objectGiven :ACTION :OBJECT)
   (#$isa :ACTION #$ReturningSomething)
   (#$giver :ACTION :SUBJECT)
   (#$givee :ACTION :OBLIQUE-OBJECT))
```

The line with the predicate `#$isa` denotes the meaning of the word in its context. The other lines show the arguments in their semantic and their syntactic (e.g. `#$objectGiven` and `:OBJECT` respectively) roles. Because users without specific linguistic backgrounds seldom understand semantic roles (as introduced by Fillmore), we determine the descriptions of them as well; ResearchCyc delivers the description in response to the following query:

```
(#$comment #$objectGiven ?COMMENT)
```

Now we fill the argument list with words from the corresponding sentence; DeNom checks every argument with which value/word it could be filled:

```
(#$arg2Isa #$objectGiven ?WHATFITS)
```

This leads to the result that an `#$objectGiven` can be every `#$SomethingExisting`. Finally, every word of the sentence is checked whether it can be used as one of the verb's arguments: DeNom checks if the word that is supposed to be used as argument (in our example `#$Pallet-TransportationConstruct` for `pallet`) is a generalization of `#$SomethingExisting`:

```
(#$genls #$Pallet-TransportationConstruct
   #$SomethingExisting)
```

If this is true, then the according word is inserted into the argument list. After checking all arguments and all words, DeNom asks the user to verify it. If the user confirms the argument list being correct, the nominalization is sufficiently specified. If one or more arguments are missing, the nominalization can be securely categorized as category 3 or 4.

Since category 3 nominalizations cannot be deemed acceptable without user interaction, we abort the classification here and present all nominalizations of category 3 and 4 to the user.

Our plan to automatically process all category 3 nominalizations is to check the broader context of the nominalization (i.e. adjacent sentences instead of adjacent phrases); also Lami's V-dictionaries could help [28], but both approaches are out of the scope of this paper. As our evaluation in the following section shows, it might not be necessary to distinguish here.

The concept presented here does not rely on domain specific ontologies. Domain specific ontologies will most likely improve the detection of acceptable nominalizations. But our current experience shows that the full specification of nominalizations heavily relies on their context within the sentence rather than on an agreed meaning given in a domain specific ontology (cf. Category 1).

Also, DeNom trusts the parser's results and we did not train the parser on requirements documents. If the parser produces faulty dependency graphs, DeNom could produce false negatives (if the parser erroneously produces a compound noun) and false positives (if the parser misses a compound noun). The evaluation shows that such errors occur but that the overall performance of DeNom is acceptable.

V. Evaluation

We evaluated DeNom with a corpus of ten specifications from Daimler AG comprising 60,000 words. The requirements specifications used in the evaluation overlap with the ones from the preparatory study but we used only SRS 4 to SRS 8 in both studies.

We compare the results from RESI using the original nominalization rule with DeNom's new nominalization checker. Therefore we inspected the specifications with RESI and DeNom in parallel, so that both tools were fed with the same documents and the same user inputs. The results of both tools were automatically logged. Then we compared the log files and manually verified DeNom's classifications. Whenever we were unsure whether a nominalization is sufficiently defined

TABLE III
RESULTS OF THE EVALUATION FOR DENOM (D) AND RESI (R). WE CALCULATED DENOM'S RECALL WITH RESPECT TO RESI'S RESULTS.

Doc.	# Words	Nom.		probl.	Precision		Recall
		R	D	Nom.	R	D	D
SRS 4	3,687	81	5	2	2%	40%	100%
SRS 5	1,598	15	2	1	7%	50%	100%
SRS 6	6,069	108	6	4	4%	67%	100%
SRS 7	12,580	246	12	3	2%	25%	60%
SRS 8	7,403	167	34	20	14%	59%	87%
SRS 9	2,923	28	4	4	14%	100%	100%
SRS 10	8,098	130	17	13	13%	76%	76%
SRS 11	2,590	57	10	7	12%	70%	100%
SRS 12	10,444	243	26	21	9%	81%	91%
SRS 13	4,094	61	13	9	15%	69%	100%
Sum	59,486	1,136	129	84	8%	65%	88%

in the sentence-wide context, we discussed our findings with experts from Daimler.

Table III details the results. The first two columns give the identifier and length of the requirements specification; columns three and four give the number of nominalizations reported by RESI (R) and DeNom (D) respectively. The fifth column gives the number of problematic nominalizations in the documents. The following two columns give the precision of both tools. The last column gives the recall of DeNom; we determined DeNom's recall with respect to RESI's results as DeNom does not identify nominalizations by itself but refines RESI's results.

In total RESI identifies 1,136 nominalizations in the corpus, 129 of which DeNom classifies as problematic. We manually inspected DeNom's results and confirmed that 84 of the 129 nominalizations are indeed problematic. DeNom increases RESI's precision from 8% to 65% on average. Unfortunately DeNom suppresses eleven nominalizations from being reported even though they are problematic.

We manually confirmed the categories of problematic nominalizations delivered by DeNom: 26 of the problematic nominalizations where category 2. DeNom does not categorize them properly due to bad English and typos.

There were also mishaps with the parser and the POS tagger that lead to errors in the processing chain. For instance, the word *current* in the phrase "After the highest current selection the selection should go to the lowest level" was processed in the meaning of present/contemporary and not in the context-correct meaning of electrical current. DeNom then wrongly decides that the nominalization "selection" is not fully specified.

Another example is the use of American and British English vocabulary: The sentence reads "The design of the wheel sensor must ensure that a tyre can be easily mounted and removed during a tyre change without damage to the wheel sensor." There, the parser does not recognize the British English spelling of "tyre" and therefore cannot produce the proper noun relationships within the sentence to ensure nominalization specification. In this case, the cross-check fails and DeNom marks the nominalization as problematic even though it is not.

There are also issues because the used ontologies are not complete. For instance, the word "implementation" is never recognized as nominalization since the ontology does not provide enough information in this specific case. We used an unmodified version of ResearchCyc. If we used DeNom in a real setting, we would simply add the missing information to avoid future mistakes. This can be done in a matter of minutes using ontology specific syntax, such as OWL (Web Ontology Language) or in this specific case CycL, ResearchCyc's internal syntax.

We also encountered some issues with the formatter and the pre-processing of exported DOORS specifications where special characters and parentheses were removed which gave problems to the parser. Most of these errors lead to the wrongful (but less problematic) detection of a problematic nominalization. This leads to false positives (worsens precision), which we consider less risky than missing an actual problematic nominalization (worse recall). Confirmed by feedback from Daimler's experts, DeNom should rather err on the safe side.

In a few cases, DeNom omitted some nominalizations that belonged to category 3 or 4. A detailed analysis of all detected and all missed nominalizations shows that eleven nominalizations have been omitted though they should have been counted as problematic. This error can be blamed on human failure: The users made mistakes during user interaction in the *CompleteProcessWords* stage.

DeNom presents 129 of the 1136 nominalizations as problematic nominalizations, nine of which are no nominalizations at all, but had been wrongly detected by RESI. The additional tests run by DeNom did not remedy this error and left the false nominalizations in the set of problematic ones.

VI. CONCLUSION

In this paper we showed that the automatic classification of potential specification flaws is possible and yields good results. The main idea was to make an academic way of automatic specification processing real-world capable by down-scaling the efforts a user has to take. We use existing technology from our tool RESI to discover the nominalizations, but extended it with our new tool DeNom to decrease the amount of user interaction and to minimize the number of false positives. Checking with real world requirements at Daimler showed that the implementation works reliably when the quality of the specification language is adequate.

A next step in refining the results is to implement the remaining check for category 3 nominalizations. For instance, a possible way to check the context could be to employ V-dictionaries. A further extension would be to not only check the glossary and adjacent words for further specifications of the nominalization, but to also include adjacent phrases and paragraphs into the context. This will blow up the search space and could increase the required user feedback again.

Concrete next steps are to assess which of the other features of RESI can be used to help the RE process and which refinements need to be made to make these features workable for users. Examples would be checking for ambiguities,

incomplete process words, etc. Another important step would be to implement DeNom in Daimler processes to gain further insight of usage and if requirements engineers (users) find the tool helpful. As of now, we do not now how a solution like RESI is perceived by industry outside of an academic context, i.e. does it really help the requirements engineers at the corresponding industry partners? Automating parts of requirements engineering might bring additional risks, which lie beyond our current horizon of expectations. For instance, users could rely too heavily on the supporting software rather than their instincts. Or maybe users feel patronized by the tools rather than accepting them as helping hands. We expect to also learn some of these psychological aspects during real world implementations of our tool.

A further step in improving DeNom beyond its current capabilities is to record and store domain specific information gathered during the process in a domain ontology or word list to further reduce user interaction. The idea is to have the program learn in its domain from the users. Then again, having a decentralized knowledge base "maintenance" could lead to wrong interpretations if the users do not use a consistent vocabulary across all projects.

A radically different approach worth investigating is using statistical methods similar to the one detailed in reference [20]. Now that we have 1,100 nominalizations categorized, we can use machine learning techniques to train a classifier. This classifier would be automatically constructed (in contrast to DeNom's hand-crafted categorization rules) and could be tuned with user feedback, e.g. when a user decides that a reported nominalization is acceptable. Such an approach would make DeNom even more self-adaptive but the version described in this paper would still be helpful for generating initial training data.

REFERENCES

[1] C. Rupp, *Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis*, 5th ed. Hanser Fachbuchverlag, Jul. 2009.

[2] S. J. Körner and T. Brumm, "Natural language specification improvement with ontologies," *Int. Journal of Semantic Computing (IJSC)*, vol. 03, no. 04, pp. 445–470, 2010.

[3] S. J. Körner, M. Landhäußer, and W. F. Tichy, "Transferring research into the real world: How to improve RE with AI in the automotive industry," in *AIRE'14: 1st Int. Workshop on Artificial Intelligence for Requirements Engineering*, Aug. 2014.

[4] N. E. Fuchs, U. Schwertel, and R. Schwitter, "Attempto controlled english — not just another logic specification language," in *8th Int. Workshop, LOPSTR'98 Manchester, UK, June 15–19, 1998 Selected Papers*, ser. Lecture Notes in Computer Science, P. Flener, Ed., vol. 1559. Springer, 1999, pp. 1–20.

[5] C. Denger, D. M. Berry, and E. Kamsties, "Higher quality requirements specifications through natural language patterns," in *Software Science, Technology and Engineering, IEEE Int. Conf. on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2003, p. 80.

[6] R. Smith, G. Avrunin, L. Clarke, and L. Osterweil, "PROPEL: An approach supporting property elucidation," in *ICSE'02: 24rd Int. Conf. on Software Engineering*, 2002, pp. 11–21.

[7] D. M. Berry, E. Kamsties, and M. M. Krieger, "From contract drafting to software specification: Linguistic sources of ambiguity - a handbook," Nov. 2003.

[8] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (EARS)," in *RE'09: 17th IEEE Int. Requirements Engineering Conf.*, Aug. 2009, pp. 317–322.

[9] Y. Pisan, "Extending requirement specifications using analogy," in *ICSE'00: 22nd Int. Conf. on Software Engineering*. Limerick, Ireland: ACM, 2000, pp. 70–76.

[10] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos, "Identifying and measuring quality in a software requirements specification," in *1st Int. Software Metrics Symposium*, May 1993, pp. 141–152.

[11] W. Wilson, L. Rosenberg, and L. Hyatt, "Automated analysis of requirement specifications," in *ICSE'97: 19th Int. Conf. on Software Engineering*, May 1997, pp. 161–171.

[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 *SEW'01: 26th Annual NASA Goddard Software Engineering Workshop*. IEEE Computer Society, 2001, pp. 97–105.

[13] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of linguistic techniques for use case analysis," in *RE'02: 10th IEEE Int. Requirements Engineering Conf.*, vol. 0, 2002, p. 157.

[14] D. M. Berry, A. Bucchiarone, S. Gnesi, G. Lami, and G. Trentanni, "A new quality model for natural language requirements specifications," in *REFSQ'06: Int. Workshop on Requirements Engineering: Foundations of Software Quality*, 2006.

[15] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[16] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski, "Software inspections: An effective verification process," *IEEE Software*, vol. 6, no. 3, pp. 31–36, May 1989.

[17] A. A. Porter, L. G. Votta, Jr., and V. R. Basili, "Comparing detection methods for software requirements inspections: A replicated experiment," *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 563–575, Jun. 1995.

[18] N. Power, "Variety and quality in requirements documentation," in *REFSQ'01: 7th Int. Workshop on Requirements Engineering : Foundation for Software Quality*, 2001, pp. 165–170.

[19] E. Kamsties and B. Paech, "Taming ambiguity in natural language requirements," in *13th Int. Conf. on Software and Systems Engineering and Applications*, Paris, France, Dec. 2000.

[20] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in *RE'06: 14th IEEE Int. Requirements Engineering Conf.* Washington, DC, USA: IEEE Computer Society, 2006, pp. 56–65.

[21] K. Verma and A. Kass, "Requirements analysis tool: A tool for automatically analyzing software requirements documents." in *Int. Semantic Web Conf.*, ser. Lecture Notes in Computer Science, A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, Eds., vol. 5318. Springer, Oct. 2008, pp. 751–763.

[22] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*, ser. Language, speech and communication. Cambridge, Mass. [u.a.]: MIT Press, 1998.

[23] Cycorp Inc., *ResearchCyc*.

[24] M. Landhäußer, S. J. Körner, and W. F. Tichy, "From requirements to UML models and back: How automatic processing of text can support requirements engineering," *Software Quality Journal*, vol. 22, no. 1, pp. 121–149, Jul. 2013.

[25] J. Krisch and F. Houdek, "The myth of bad passive voice and weak words – an empirical investigation in the automotive industry," in *RE'15: 23rd IEEE Int. Requirements Engineering Conf.*, Aug. 2015.

[26] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, pp. 207–239, 2008.

[27] M.-C. de Marneffe and C. D. Manning, "Stanford typed dependencies manual," Tech. Rep., Feb. 2015.

[28] G. Lami, S. Gnesi, F. Fabbrini, M. Fusani, and G. Trentanni, "An automatic tool for the analysis of natural language requirements," Pisa, Italia, Tech. Rep. 2004-TR-40, Sep. 2004.

[29] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, vol. 16, no. 3, pp. 365–395, Oct. 2010.