Quality Assessment for Requirements based on Natural Language Processing

Mathias Soeken[1,2], Nabila Abdessaied[1], Arman Allahyari-Abhari[1,2],
Andi Buzo[3], Liana Musat[3], Georg Pelz[3], and Rolf Drechsler[1,2]
[1] Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[2] Department of Mathematics and Computer Science, University of Bremen, Germany
[3] Infineon Technologies AG, Neubiberg, Germany
msoeken@cs.uni-bremen.de

***Abstract*—Recently, many approaches for automatic information extraction from technical specifications in the area of electronic design automation have been proposed. For this purpose, techniques from natural language processing are used. In order to lower the bars for designers and customers, some approaches do not intend to restrict the natural language that is used to describe the specifications. However, ambiguity and vagueness in the language often cause wrong or bad results obtained from the algorithms. This work describes preliminary ideas for automatic approaches that assist in writing the specification and aim at increasing the quality of the text. Besides improved comprehensibility, better written specifications will also enhance the quality of the automatic extraction approaches in subsequent steps of the design flow.**

I. Introduction

The quality of requirements in textual specification documents has a significant impact on the final product. Requirements of a low quality can lead to misunderstandings and therefore to errors in the design flow that are usually difficult to detect or detected too late. Consequently, deadlines must be postponed which results in an overall higher cost of production. Furthermore, badly written requirements also impede the application of automatic methods for requirement formalization. In this paper, we present two approaches that check the quality of requirements, the first one is based on static syntactic and semantic analysis whereas the second one is based on requirement guidelines.

The first approach makes use of syntactic and semantic properties of the sentence. Basis for the quality measure are e.g. the possible interpretations of a sentence which corresponds to the parses of a sentence and the possible meanings of a word in a sentence which can be determined using dictionaries such as WordNet [1].

The second approach considers requirement guidelines of which several exist and aid designers in writing good requirements. These guidelines are either provided globally to a large audience e.g. by means of books or they are used internally as an agreement between employees of a company. Typical examples for rules defined in such guidelines are the avoidance of imprecise words such as “should” or “could”, adjectives such as “high”, “robust”, or “low enough”, or the use of passive verb forms. If many of those rules have been defined, it becomes cumbersome to manually check whether all of them are followed. We present algorithms based on natural language processing techniques that for a given requirement can automatically determine whether a rule has been violated.

Besides leading to more comprehensive specifications, the proposed algorithms are also of significant interest to information extraction algorithms that have recently been proposed in the field of electronic design automation [2], [3], [4]. Algorithms for checking the quality of requirements have been proposed in the past. As one example, NASA developed the tool “*Requirements Assistant*” for internal requirements quality assurance [5]. It helps to ensure that natural language requirements are complete, consistent, feasible, and unambiguous. Similar tools have been presented in [6], [7], however, non of them are freely available. Other related work has also previously discussed metrics to determine the linguistic quality of texts, sentences, and words [8], [9].

Our algorithms are implemented using techniques from *Natural Language Processing* (NLP). To evaluate our approach we collected and manually annotated requirements that are used in industrial specifications.

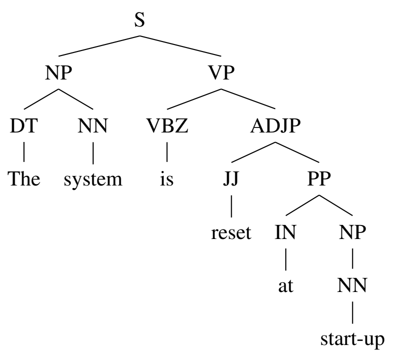
II. Preliminaries

This section describes the NLP techniques that are used for the implementation of the proposed algorithms. A good overview on NLP techniques can be found in (e.g. [10], [11]).

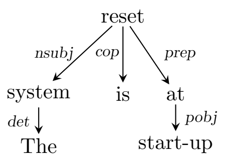
A. Phrase Structure Trees

A *Phrase Structure Tree* (PST) is a tree containing structural information about a sentence. The root node represents the whole sentence. The non-terminal nodes represent the syntactic grammar structure in terms of constituents, while the terminal nodes are the atomic words of the sentence. The analysis of the sentence and annotation into a PST is performed by structural parsers such as the one contained in the Stanford CoreNLP natural language processing toolkit [12].

Due to ambiguities of natural language, there are in fact many possible PSTs for one given sentence. How the PSTs are generated depends on the structural grammar the parser uses. The parser of our choice uses a *Probabilistic Context-Free Grammar* (PCFG, [10]) as back-end grammar for the parse tree computation. Any possible PST of a sentence is assigned a score indicating the probability to be the correct parse. The



(a) Phrase structure tree



(b) Dependency graph

Fig. 1. Common data structures in natural language processing

PST with the highest PCFG score is the one which is most likely to be correct.

Example 1: A phrase structure tree for the sentence "The system is reset at start-up." is depicted in Fig. 1(a).

B. Dependency Graphs

In order to represent dependencies between individual words, NLP techniques make use of dependency parses [13]. For this purpose, binary relations describing syntax and semantic are extracted from a sentence. A dependency is given as $r(g, d)$ with a relation r, a governor g, and a dependent d. As an example the relation *nsubj* binds a verb to its subject. Other relations are *nn* that groups compound nouns or *det* that assigns a noun to its determiner. In [13], altogether 48 relations have been arranged in a grammatical relation hierarchy. Given a sentence s, a dependency graph is an edge-labeled directed graph in which vertices represent words of s. There is an edge $g \xrightarrow{r} d$ between two different words g and d if and only if $r(g, d)$ is a dependency in s.

Example 2: The dependency graph for the sentence "The system is reset at start-up." is depicted in Fig. 1(b).

C. WordNet

WordNet [1], developed by linguists and computer scientists at Princeton University, is a large lexical database of English that is designed for use under program control. It groups nouns, verbs, adjectives, and adverbs into sets of cognitive synonyms called *synsets*, each representing a lexicalized concept. Each word in the database can have several *senses* that describe different meanings of the word. In total, WordNet consists of more than 90,000 different word senses, and more than 166,000 pairs that connect different senses with a semantic meaning.

Further, each sense is assigned a short description text which makes the precise meaning of the word in that context obvious. Frequency counts provide an indication of how often the word is used in common practice. The database does not only distinguish between the word forms noun, verb, adjective, and adverb, but further categorizes each word into sub-domains. Those categories are e.g. *artifact*, *person*, or *quantity* for a noun.

III. Static Sentence Analysis

This section describes a static algorithm, which determines a single quality measure to a given sentence. For this purpose, we make use of established syntactic and semantic analysis methods from NLP. More precisely, we are trying to solve the following problem:

Problem 1 (Sentence quality): Given an English sentence, the *sentence quality* problem asks to determine a quality measure indicating whether the sentence is *good*, *medium*, or *bad* in terms of comprehension.

The approach is not domain-specific and is designed to handle any English sentence. The syntactic and semantic quality measures of a sentence are determined separately. Therefore, first both computations are briefly described. Subsequently, the outcome is consolidated into a single value indicating the overall quality of the input sentence.

A. Syntactic Quality

In this work the syntactic quality of a sentence is considered directly ambivalent to the number of structural ambiguities. Therefore, the smaller the number of structural ambiguities is, the better is the syntactic quality. There are lots of ways to analyze the structure of a sentence. The basic underlying data structure for the computations are phrase structure trees. As a basis for the computation of the syntactic ambiguities we use the PST with the highest PCFG score and compare the structural differences to the next best 100 parses. Based on these PSTs, the computation of the syntactic quality is computed by the following steps:

1) *Isomorphic subtrees:* The most likely parse tree is compared to each of the next best 100 parse trees. The algorithm computes the difference of any parse tree to the best parse tree in terms of its subtrees. The average of these subtree matchings over all 100 next best parses referred to as *Isomorphic Subtree Ratio* (ISR) and can be assigned a value between 0 and 1. The lower this ratio is, the more structural ambiguities the sentence contains. This is the main indicator for syntactic quality.
2) *Sentence length penalty:* The longer a sentence, the more likely it is that there are less structural ambiguities. Therefore, in a last step the syntactic quality, computed over the ISR and the score, is decreased afterwards by a value according to the amount of words in the sentence.

B. Semantic Quality

The semantic quality of a sentence is determined by its amount of semantic ambiguities. WordNet is used to determine these. We consider a word ambiguous if it has more than one synset in the WordNet dictionary. The computation of the semantic quality is simple enough for an exact description. It is done in the following way:

Let n be the number of nonambiguous words and m the number of ambiguous words of a given sentence. Then, $a = n/(n+m)$ describes the distinct portion and $b = m/(n+m)$ the ambiguous portion of the sentence. Note that $a + b = 1$,

TABLE I
COMPARISON OF THE ALGORITHM AND A SUBJECTIVE OPINION

Quality predicate	**Algo.**	**Subj.**	**Matches**	**Misses dist. 1**	**Misses dist. 2**	**Matching percentage**
good	32	26	20	8	4	62.5 %
medium	53	41	29	24	×	54.7 %
bad	37	55	31	4	2	83.8 %
total	122	122	80	36	6	65.6 %

and therefore the sentence is free of ambiguities if $a = 1$. In order to formalize a semantic quality measure we further want to take the number of meanings for each ambiguous word into account. Let k_i be the number of different synsets of the i-th ambiguous word, then the basic semantic quality is given by the formula

$$q_{\text{sem}} = a + b \cdot \frac{m}{\sum_{i=1}^{m} k_i}. \tag{1}$$

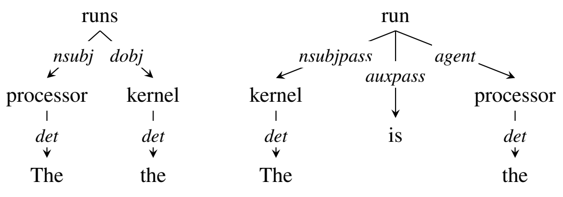
After the computation over ambiguous words, we need to consider compounds, which are usually nominal compounds such as “header file”. If a compound is detected, so if any word chain in the ambiguous portion of the sentence has at least one meaning in WordNet, the semantic quality is modified. The sum of the meanings of every single word in the compound is simply substracted from the overall sum of meanings and then replaced by the number of meanings of the compound covering these words. Therefore, detecting nominal compounds in a sentence usually reduces the overall number of ambiguities and thus increases the semantic quality.

C. Experimental Evaluation

Experiments showed that a 40:60 ratio of syntax to semantics seems to be a good general configuration for the overall sentence quality of the algorithm. To be able to compare the result to a human’s subjective opinion, the measured sentence quality is transfered into an according quality predicate *good*, *medium*, or *bad*. We applied the algorithm to a test set of requirements that have been extracted from various specifications [14], [15], [16]. The test set contains 103 different requirements given by a total of 122 sentences. For comparison the requirements were split into their single sentences. Table I provides a comparison of the quality evaluation of the algorithm and a subjective opinion. Note that in average the algorithm assigned a better quality predicate than the subjective opinion. The numbers of matches and misses between both evaluations are depicted in the next columns, while the number of misses is distinguished by the distance of the quality predicates of subjective opinion and algorithm (the distance between *good* and *bad* is 2, whereas their distance to *medium* is 1). The last column gives an overview of the matching percentages for every quality predicate with an overall matching of 65.6 %.

IV. GUIDELINE VALIDATION

Several guidelines exist which aid designers in writing good requirements. These guidelines are either provided globally to a large audience e.g. by means of books or they are used internally as an agreement between employees of a company and their customers. We aim at providing solutions to the following problem:



(a) Active voice (b) Passive voice

Fig. 2. Active and passive voice

Problem 2 (Guideline checking): Given a set of rules from guidelines how to write requirements and a natural language requirement R, the *guideline checking* problem asks whether R adheres to the rules.

We propose to solve the problem using natural language processing techniques. For our experimental evaluation we have composed a set of such rules that we extracted from several guideline documents [17], [18], [19]:

R1. Define one requirement at a time.
R2. Avoid conjunctions (and, or, with, also) that make multiple requirements.
R3. Use simple direct sentences.
R4. Each requirement must contain a subject and a predicate.
R5. Avoid let-out clauses (unless, except, if necessary, but, when, unless, although).
R6. Avoid expressing suggestions or possibilities (might, may, could, ought, should, could, perhaps, probably).
R7. Avoid weak phrases and undefined terms (adequate, as a minimum, as applicable, easy, as appropriate, be able to, be capable, but not limited to, capability of, capability to, effective, if practical, normal, provide for, timely, tbd, user-friendly, versatile, robust, approximately, minimal impact, etc., and so on, flexible, to the maximum extent, as much as possible, minimal impact, in one whack, different, various, many, some of, diverse)
R8. Do not speculate (usually, generally, often, normally, typically).
R9. Avoid wishful thinking (100% reliable, safe, handle all failures, fully upgradeable, run on all platforms).
R10. Define verifiable criteria.

We have taken the rules as they were written in the original documents. It is debatable whether all these rules make sense in each context, but it can clearly be seen, that most of the rules were not postulated with having automatic approaches in mind. As an example rule R10 “*Define verifiable criteria.*” is very difficult to be checked automatically.

In order to handle this vagueness, the decision of the algorithms is given in terms of a tri-state value. This value distinguishes the cases of whether a rule is clearly violated or not, or whether no confident result could be computed.

Consider e.g. R3 “*Use simple direct sentences.*” One heuristic to use is to check for active and passive voice in sentence.

TABLE II
EXPERIMENTAL RESULTS

Rules	Manual Class.		Auto. Class.		Classifier Evaluation					
	T	F	T	F	SA	TP	TN	FP	FN	Acc.
R1	62	41	20	83	51	15	84	47	5	49.51%
R2	88	15	98	5	85	84	2	4	14	82.52%
R3	84	19	103	0	84	84	0	0	19	81.55%
R4	90	13	95	8	84	83	2	7	12	81.55%
R5	102	1	96	7	97	96	8	6	0	94.17%
R6	94	9	96	7	101	94	8	0	2	98.06%
R7	92	11	88	15	95	86	13	6	2	92.23%
R8	102	1	103	0	102	102	0	0	1	99.03%
R9	103	0	103	0	103	103	0	0	0	100.00%
R10	9	7	6	10	13	6	11	3	0	81.25%
Total	826	117	808	135	815	753	128	73	55	86.43%

If the sentence is given in passive voice, we can determine that the rule has been violated. Note that the contrary is not necessarily true. By making use of typed dependencies it can easily be checked whether a sentence is given in active or passive voice, since different relations are found in the corresponding typed dependency graphs. While the subject is indicated as the dependent of an '*nsubj*' relation in a sentence in active voice, the relation will be '*nsubjpass*' when using passive voice (cf. Fig. 2). But it cannot only be checked rather easy whether the rule is violated by inspecting if such dependency relations occur in the sentence; passive sentences can also be translated automatically using NLP techniques.[1]

A. *Experimental Evaluation*

The proposed approach has been implemented in Scala based on the Stanford CoreNLP library. We evaluated our approach using the same set of requirements as for the static approach in the previous section and manually annotated them according to the given rules. The manual annotations were then compared to the results of the classifier.

Table II summarizes the obtained results for the conducted experiments. The first column gives the rules as defined in this section. In the following columns, the respective annotations for the manual classification (Manual Class.), the results for the automated classification (Auto. Class.), and the concluded results for the classifier evaluation are presented. T, F, SA, TP, FP, TN, and FN refer to the number of *true annotated*, *false annotated*, *same annotated*, *true positives*, *false positives*, *true negatives*, and *false negatives*, respectively. The last column represents the accuracy (Acc.) computed by $\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$.

The accuracy of each rule (except R1) is higher than 80% and reaches 99% and 100% for *R8* and *R9*, respectively. Our rule based tool has a significant performance in general since it has an average accuracy of 86.43%.

V. CONCLUSIONS

We have presented two automatic approaches that assist the designer in writing better requirements in specifications by (i) checking syntactic and semantic properties of the requirement, and (ii) validating the requirement with respect to rules from a guideline specification. For a selected set of typical rules it has been shown that our methods work effectively. For future work a thorough case study should further evaluate the practicability of our approaches. Also, it should be investigated which rules are suitable for automatic fixing.

[1]This is e.g. being illustrated using the *Voice Conjugator* widget at www.contextors.com.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project SPECifIC under grant no. 01IW13001 as well as by the German Research Foundation (DFG) within a *Reinhart Koselleck* project under grant no. DR 287/23-1.

REFERENCES

[1] G. A. Miller, "WordNet: a lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
[2] I. G. Harris, "Extracting design information from natural language specifications," in *DAC*, 2012, pp. 1256–1257.
[3] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *TOOLS*, 2012, pp. 269–287.
[4] M. Soeken, C. B. Harris, N. Abdessaied, I. G. Harris, and R. Drechsler, "Automating the translation of assertions using natural language processing techniques," in *Forum on Specification & Design Languages*, 2014.
[5] S. H. Consultancy. Requirements assistant. Available at http://www.requirementsassistant.nl/.
[6] G. Lami, "Quars: A tool for analyzing requirements," DTIC Document, Tech. Rep., 2005.
[7] ClearSpecs. Tekchecker. Available at http://clearspecs.com.
[8] 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.
[9] A. C. Graesser, D. S. Mcnamara, and J. M. Kulikowich, "Coh-metrix: Providing multilevel analysis of text characteristics," in *Educational Researcher*, vol. 40, no. 5, 2011, pp. 223–234.
[10] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Pearson Prentice Hall, 2008.
[11] N. Indurkhya and F. J. Damerau, *Handbook of Natural Language Processing*, 2nd ed. Chapman & Hall/CRC, 2010.
[12] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *ACL: System Demonstrations*, 2014, pp. 55–60.
[13] M.-C. de Marneffe, B. MacCartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," in *Conf. on Language Resources and Evaluation*, 2006, pp. 449–454.
[14] P. R. Harvey. (2010) (NASA) flight software requirements. Available at ftp://apollo.ssl.berkeley.edu/pub/RBSP/1.1.%20Management/5%20 Meetings/PhaseB_080902_IPDR/Documents/ RBSP_EFW_FSW_002_Requirements.pdf.
[15] Intel. (2010) Intel® active management technology (Intel® AMT) 7.0 release : Fw & sw product requirements document (PRD). Available at http://www.intel.de/content/dam/www/public/us/en/documents/product-specifications/amt-7-0-release-fw-sw-prd.pdf.
[16] T. Morgan. (2003) Requirements and functional specification : Evla correlator backend. Taken from National Radio Astronomy Observatory, available at http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/BE_rfs_1.pdf.
[17] I. F. Alexander and R. Stevens, *Writing better requirements*. Pearson Education, 2002.
[18] IBM. (2009) Get it right the first time: Writing better requirements. Available at http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/topic/com.ibm.help .download.doors.doc/pdf92/get_it_right_the_first_time.pdf.
[19] W. Wilson, "Writing effective requirements specifications," in *Software Technology Conference*, 1997, available at NASA Software Assurance Technology Center (SATC) http://www.csc.kth.se/utbildning/kth/kurser/DD1363/NASARequirements .html.