Generating a Linguistic Model for Requirement Quality Analysis

Juyeon Kang
PROMETIL
42 avenue du Général de Croutte
31100 Toulouse, France
j.kang@prometil.com

Jungyeul Park
Department of Linguistics
University of Arizona
Tucson, AZ 85721
jungyeul@email.arizona.edu

Abstract

In this work, we aim at identifying potential problems of ambiguity, completeness, conformity, singularity and readability in system and software requirements specifications. Those problems arise particularly when they are written in Natural Language. We describe them from linguistic point of view but the business impacts of each potential error will be considered in system engineering context where our corpus come from. Several standards give the criteria on writing good requirements to guide requirement authors. These properties are linguistically observable because they appear as lexical, syntactic, semantic and discursive problems in documents. We investigate error patterns heavily used, by analyzing manually the corpus. This analysis is based on the requirements grammar that we developed in this work. We then propose an approach to identify them automatically by applying the rules developed from the error patterns to the POS tagged and parsed corpus. By using error annotated corpus, we can train the error model using CRFs and evaluate it. We obtain overall 79.17% F_1 score for the error label annotation task.

1 Introduction and Context

In order that a system is realized and become operational in real applications, it follows several stages of conception, development, production, use, support and retirement (ISO/IEC TR 24748-1, 2010). During the concept stage, we identify and document the stakeholder's needs in the system requirements specification (Hull et al., 2011). Writing clearly all required elements without ambiguities (Berry et al., 2003) in the specifications is an essential task before passing to the development stage (Galin, 2003; Bourque et al., 2004). According to the *2015 Chaos report* by the Standish Group[1], only 29% of projects was successful[2]. And 50% of the challenged projects is related to the errors from the Requirement Engineering and 70% of them comes from the difficulties of understanding of implicit requirements. All these errors do not lead to the failure, but generate useless information. It is well known that the costs to fix errors increase much more after that the product is built than it would if the requirements errors were discovered during the requirements phase of a project (Glas, 2002; Stecklein et al., 2004).

However, when writing or revising a set of requirements, or any technical document, it is particularly challenging to make sure that texts read easily and are unambiguous for any domain actor (Weiss, 1990; Grady, 2013). The previous experience shows that even with several levels of proofreading and validation, most texts still contain a large number of language errors (lexical, grammatical, semantic, style, etc.), and lack of overall cohesion and coherence. Risks emerge from poorly written texts, and from various forms of incoherence. For example, *Progressively heat the probe X27* relies too much on the operators knowledge and practice: what temperature should be reached and in how much time? A

[1] http://www.standishgroup.com

[2] They studied 50,000 projects around the world, ranging from tiny enhancements to massive systems re-engineering implementations.

wrong interpretation may lead to accidents and damages.

Tools controlling the authoring quality of requirements have been developed in the past with the use of templates or boilerplates meant to guide the technical writer. This is most notably the case for the RAT-RQA system[3] and of the RUBRIC system (Arora et al., 2013). Let us also cite two major CNL-based prototypes which are of much interest for requirement authoring: ACE (Fuchs, 2012), which stands for Attempto Controlled English. This system makes an in-depth language semantic analysis. It was initially designed to control software specifications, and has been used more recently in the semantic web. PENG (Processable English) (White and Schwitter, 2009) is a computer-processable controlled natural language system designed for writing unambiguous and precise specifications. These systems make heavy use of syntactic analysis, which is rather costly. A synthesis of CNL based systems is developed in (Kuhn, 2013).

We also find the systems of requirements quality analysis based on the shallow parsing techniques. First, SEMIOS system[4] is relevant for requirements where the language is complex and sometimes ill-formed. It detects several types of errors, lexical, syntactic and related to style. Error detection in this system depends on the discourse structure analysis. Second, (Berrocal Rojas and Barrantes Sliesarieva, 2010) proposes a software prototype for controlling if a requirement satisfies the criteria of the high quality requirement defined in DO-178b. This work focuses on the detection of inaccurate, non-verifiable and ambiguous elements in requirements by means of lexical (using WordNet and VerbNet) and syntactic analysis.

There are other interesting approaches and tools developed for automatically analyzing the requirements specifications but we do not develop all of them in this paper because of the lack of space. We invite the readers to consult (Gnesi et al., 2005; Fabbrini et al., 2001; Zapata Jaramillo, 2010).

The model that we propose identifies the potential errors in natural language requirements by applying error patterns rules to the POS tagged and syntactically parsed sentences. It depends on the requirements grammar that we elaborate from the requirements authoring guidelines. In §2, we introduce the essential constraints of authoring high quality requirements with examples, and in §3, develop the requirements grammar rules corresponding to each constraint. Errors patterns are also described in this section as they are induced by verifying if requirements are correctly written following the rules of the requirements grammar. The §4 describes the methods and results of our experiments elaborated for generating an adapted model of automatic error patterns labeling to requirements documents. The main contribution of the paper is as follows: (1) We define requirements grammar and their error patterns. (2) We create training and evaluation data for building an error pattern model and assigning error labels. To the best of the author's knowledge, it is the first time to achieve such results by using the automatically learned model from the training data set. It would be suitable for the general purpose error annotation for requirements authoring quality.

2 Requirements Authoring and Quality

Among technical documents, requirements are a central issue since they must comply with a high number of constraints of e.g. readability, lack of ambiguity and implicit data, feasibility, relevance, traceability, conformity and overall cohesion and coherence (Firesmith, 2003; Alred et al., 2012). The principles of the authoring quality of requirements are defined in different standards like IEEE 830-1998 (IEEE Recommended Practice for Software Requirements Specification), ISO/IEC/IEEE29148:2011 (Systems and software engineering – Life cycle processes – Requirements engineering), ARP4754A (Aerospace Recommended Practice) and also in the recommendations of INCOSE (Guide for Writing Requirements), IREB (International Requirements Engineering Board) and the controlled natural languages (e.g. ASD-STE 100[5]). The authoring constraints specify the syntax, the semantic along with the style and the lexical items that the technical authors must respect.

[3] http://www.reusecompany.com

[4] http://www.semiosapp.com

[5] Simplified Technical English, Aerospace and Defense, by Industries Association of Europe, Issue 5, 2010

In this paper, we focus on the five constraints (ambiguity, conformity, completeness, singularity, readability), considered as being the most critical by requirements authors, with examples and descriptions. All examples was extracted from our test corpus. It contains technical requirements and some of them are anonymized because of confidential problems. But they remains meaningful enough to show real problems in requirements texts.

We follow mainly the definitions of these constraints, proposed by the above mentioned standards and guidelines of IEEE and INCOSE.

2.1 Non-Ambiguity

An ambiguous term can convey several information which lead the requirement to different interpretations of what the system is expected to do. A requirement must be interpreted in only one way without ambiguities. The following examples contain a lexical ambiguity in (Req1) with the fuzzy adjective *standard* and a grammatical ambiguity in (Req2) with the combinator *or*.

- (Req1) *The maximum pressure loads at the standard operating temperature shall be 6.*
- (Req2) *The CPU system shall set these signals in output **or** shall send them directly to the platform.*

2.2 Conformity

A requirement must be written conforming to the standard structure and style defined by a company or a group of authors. Not respecting this standard increases the problem of understanding and makes difficult to identify the main requirements from the other types of sentences having a similar structure like procedures, instructions, recommendation, etc. The use of *should* instead of *shall* in (Req3) makes the requirement non mandatory.

- (Req3) *Paint coatings should also assist in the overall maintenance of the vehicle by providing easy to clean surfaces.*

2.3 Completeness

A requirement must contain complete information in itself without needing extra elements to understand correctly the requirement. In the example (Req4), the requirement missed the agents who shall realize the actions, and in (Req5), *these* and *this* refer to some elements which can be identified with extra contextual information.

- (Req4) *In particular the received configurations shall be used and the communication signal shall be isolated.*
- (Req5) *If these systems are required for safety purpose, this requirement shall not prohibit the use of the supply systems.*

2.4 Singularity

A singular requirement must express a single idea and characteristic concerning what the system has to make. The (Req6) contains multiple actions *shall deliver* and *reload*, introduced by the use of the combinator *and*. The (Req7) expresses the main action, then justify why this action is required (*in order to...*). This last is not a part of the requirement.

- (Req6) *The system shall deliver data and reload the configuration checks performed and not performed.*
- (Req7) *Seats shall be selected at the discretion of each customer in order to [accommodate differences in operations and passenger preferences].*

2.5 Readability

A complex requirement makes difficult the comprehension on given requirements and increase the cognitive works of the reader. In (Req8), the quantifier *all* needs to be specified by a list to be easily readable. In (Req9), the three acronyms should be defined in a glossary and if not, the requirement will not be understandable without specific domain knowledge.

- (Req8) *All exterior graphics shall be applied to the vehicle in accordance with Customer specifications.*
- (Req9) *Static RAM or dynamic EPROM windows shall be covered with labels that are opaque at the UV erasing wavelengths.*

The above requirements (Req1)∼(Req9) illustrate the counterexamples of high quality requirements,

which do not respect the non-ambiguity, conformity, completeness, singularity and readability. Some of them will be reconsidered in §3.1 to describe the Requirements Grammar.

3 Requirements grammar and Errors patterns

As shown in §2, the requirements texts need to be qualified as unambiguous, conforming, complete, readable and singular. Such constraint of a requirement written in Natural Language form a specific linguistic genre that we call "requirements grammar". We consider in this work the five previously explained constraints for writing good requirements and define the corresponding rules in our requirements grammar. We also elaborate the types of most frequent language errors as errors patterns, and describe them in relation with the rules of requirements grammar. Table 1 describes a list of error patterns that we developed based on requirements grammar.

3.1 Ambiguity rules

Rule 1: A requirement should not contain ambiguous adjectives, adverbs, verbs and nouns which can lead it to several interpretations, such as *significant, flexible, sufficient, adequate, nearly, correctly, properly, minimize, optimize, malfunction, undesirable effects*, etc. All adverbs ending in *-ly* particularly make requirements unverifiable. These terms can be replaced or complemented by a value, a set of values or an interval. The example (Req1) shows the case that prohibits this rule. If the *standard operating temperature* is not defined in the text, it should be reformulated like *the standard operating temperature between 5°C and 10°C*.

Rule 2: A requirement should avoid the use of the combinator *or* and the combination of *and* and *or*. The conjunctions *or* and *and*, which coordinate two actions verbs and two subjects, are not acceptable as it raises a critical ambiguity problem (if they appear in a main clause, it is more critical than in a subordinated clause). The example (Req2) shows the case that the *or* is used between two main action verbs *shall set* and *shall send*.

3.2 Conformity rules

Rule 3: A requirement expresses an obligation that states what the system should realize. The modal *shall* is mainly used for mandatory requirements. Other modals verbs like *must, should, could, would, can, will, may, should* are not allowed in writing the main action of a requirement. The example (Req3) uses the modal verb *should* which expresses a recommendation rather than an obligation. It means that *it is recommended that Paint coatings assist in overall maintenance but not obligatorily*. The impact of not respecting this requirement can be critical for the system.

Rule 4: The negation markers should be avoided in a main clause as they states what the system does not do as we can see in *The system shall not transfer unauthorized data to the sub-systems*. This requirement should be reformulated like *The system shall transfer only authorized data to the sub-system*.

3.3 Completeness rules

Rule 5: A requirement should be written in the active voice because the majority of passive sentences do not include explicit agents to indicate exactly who perform the action. For example, in the requirement *It shall be tested in the following conditions:...*, we need extra information to identify what will be tested and who will test. The example (Req4) shows the same situation.

Rule 6: Referential ambiguities appear when the demonstrative and possessive pronouns like *it, they, them, their...* are used with unclear antecedents in requirements. These terms can refer to more than one element of the same sentence or of the previous sentence. In the example (Req5), the pronoun *these* probably refers to some elements introduced in the antecedent requirements. All elements that *these systems* refer to should be clearly specified in the given requirement.

3.4 Singularity rules

Rule 7: A requirement should express only one action and one idea (one subject) in a requirement. First, we find two types of erroneous structures introducing multiple actions in a requirement: 1) more than two action verbs enumerated in a list, 2) more than two action verbs coordinated by more than one *and*. Second, when the subject is expressed in using *and* like *X and Y shall...*, we consider the requirement as having multiple subjects and multiple thoughts.

The example (Req6) describe the performance of several actions and ideas in a requirement. In this kind of case, when one of the actions is updated, it can influence on the other action, consequently makes difficult the maintenance and validation of the requirement.

Rule 8: A requirement should contain appropriate information, not including the solution and the purpose of the given requirement. These extra information should be presented separately in another documents.

In the example (Req7), *in order to* introduces a new information: the reason why *seats are selected at the discretion of each customer*. The author should not include the justification part in the requirement. This problem often occurs using the following structures: *to, so as to, for the purpose of, so that, in order that*, etc.

3.5 Readability rules

Rule 9: The use of universal quantifiers like *every, all, each, several, a, some,* etc. should be avoided because they generates the scope ambiguity. For example, in the requirement *All sub-systems shall have their fire alarm*, the quantifier *all* does confuse readers if the meaning is that *all sub-systems share one alarm* or *all sub-systems has its own alarm*.

Rule 10: In principle, a requirement text should have available a glossary where the acronyms and abbreviations are defined in order to help the reader to understand the concepts related to them. Otherwise an acronym should have a definition inside of the requirement like *The APU (Auxiliary Power Unit) system*.

4 Experiments and Results

4.1 Building training data

We build training data by using POS tagging and syntactic parsing. We define five types of errors as described in §2: ambiguity (`AMBI`), conformity (`CONF`), completeness (`COMP`), singularity (`SING`), and readability (`READ`) and we write heuristic rules based on error patterns. We use the IOB format for error labels, in which `B-` for 'beginning' of the label, `I-` for 'inside', and `O` for 'outside': *e.g.* `B-AMBI` and `I-AMBI` for the beginning and the inside of the ambiguity label. Since we use the automatic method to build our training data, we want to minimize the error rate in our data. Therefore, we introduce the filtering method by using two different algorithms to filter out instances that we consider as errors for POS tagging and syntactic analysis. To so do, we simply use the consensus filtering method by the intersection operation as follows:

$$\hat{\mathcal{D}} = \mathcal{D}(\mathcal{M}_1) \cap \mathcal{D}(\mathcal{M}_2) \quad (1)$$

where $\mathcal{D}$ is raw text data, $\mathcal{M}_i$ is a learning algorithm to annotate raw text data, and $\hat{\mathcal{D}}$ is filtered annotated data. For POS tagging, we use a hidden Markov model (HMM) and conditional random fields (CRFs) in which we trained with POS information of English treebank data[6]. We use a TnT tagger (Brants, 2000) and Wapiti described in (Lavergne et al., 2010) for the HMM and CRF annotation, respectively. For syntactic parsing, we use two pretrained dependency parsing models for MaltParser (Nivre et al., 2006).[7] We use syntactic parsing results to detect the correct range of contaminator errors described in §3.1, in which `or` and `X'` are dependent of `X` in `X or X'` (for the `AMBI` error label). The length of `X` and `X'` can vary in the sentence and it would be difficult to detect them without syntactic analysis. Otherwise, error annotation rules are entirely based on POS tagging and lexical patterns and these rules are described in detail throughout §2 and §3. For raw text data, we use ukWaC (the largest English web-crawled resource), one of the WaCky corpora presented in (Baroni et al., 2009). We believe that ukWaC contains large numbers of texts written in technical English as it shows some lexical and structural similarity to requirements texts: use of the modal *shall*, action verbs, terms expressing needs, etc. Finally, after consensus between POS tagging and syntactic analysis, we obtain 82,847 sentences with 844,770 tokens for the training data set. For the error annotation based on heuristic rules, we give priorities for certain error patterns. Therefore, when there are several possibilities to annotate errors in the same word, we use the following error precedence:

[6] `https://catalog.ldc.upenn.edu/LDC99T42`

[7] `http://www.maltparser.org/mco/english_parser/engmalt.html`

Error patterns	Description	Impacts and rules
Combinators	This error pattern related to the use of combinators *or* and *and* concerns the rules 2 and 7, respectively. • `X or X'` where POSs (or phrase type) of X and X' are same. • `X and/or X'` where POSs (or phrase type) of X and X' are same. • `X and X'` where POSs (or phrase type) of X and X' are same. X= verb (infinitive form), verb phrase, noun, noun phrase, adjective, value followed by a unit of measurement	Ambiguity 2 Singularity 7
Pronouns	This error pattern related to the use of possessive and demonstrative pronouns concerns the rule 6. • `Pron(possessive), Noun`: *their application* • `Pron(possessive), NP`: *their proper development* • `Pron(demonstrative), modal(shall)`: *this shall*, *these shall*	Completeness 6
Lexicals	This lexical error pattern concerns the rules 1, 8, 9 and 10 of §3. Requirements containing one of the following lexical items: ambiguous terms (Rule 1), purpose expressions (Rule 8), quantifiers (Rule 9), acronyms (Rule 10), raises the problem of ambiguity, singularity and readability. Due to lack of space, we do not give its complete lists but the main items are mentioned in each rule.	Ambiguity 1 Singularity 8 Readability 9, 10
Passive construction	This error pattern related to the use of passive construction concerns the rule 5. We do not consider the passive construction followed by the preposition *by* which introduces the agent as being erroneous. • `modal(shall), be, AdvP, Verb(Action, PP)`: *shall be used*, *shall be properly used*	Completeness 5
Negations	This error pattern related to the negation marker concerns the rule 4. `modal` is only the mandatory modal *shall*. • `modal, Neg`: *shall not*	Conformity 4
Modals	This error pattern related to the use of different types of modal verbs concerns the rule 3. `modal` excepts the mandatory modal *shall*. • `modal, AdvP, Verb(Action, Inf)`: *would implement*, *should correctly implement*	Conformity 3

Table 1: Correspondence between errors patterns, impacts and rules

The	DT	O
analytes	NNS	B-AMBI
or	CC	I-AMBI
investigations	NNS	I-AMBI
covered	VBN	O
by	IN	O
the	DT	O
Scheme	NNP	O
shall	MD	B-COMP
be	VB	I-COMP
selected	VBN	I-COMP
on	IN	O
the	DT	O
basis	NN	O
of	IN	O
their	PRP$	B-COMP
clinical	JJ	I-COMP
relevance	NN	I-COMP
.	.	O

Figure 1: An example sentence from training data: *analytes or investigations* represents the ambiguity error, *shall be selected* is annotated as completeness error, and *their clinical relevance* has the completeness error.

$$\text{combinators} > \text{pronouns} > \text{lexical} > \text{passive construction} > \text{negations} > \text{modals} \quad (2)$$

Figure 1 shows an example sentence from our training data. In this figure, first, *analytes or investigations* represents the ambiguity error because of *or* as explained in the Rule 2 of §3.1. Second, *shall be selected* is annotated as completeness error because the information about who realize the required action is not specified as shown in the Rule 5 of §3.3. Third, *their clinical relevance* also has the completeness error because of the possessive pronoun *their*. It probably refers to one of the following antecedents: *analystes*, *investigations*, *the Scheme* but we need extra information to correctly identify the reference of *their* (see the Rule 6 of §3.3).

We use CRFs for training. Since we are heavily based on lexical and POS information, we use a simple feature set, in which ± 2 word/POS window context, and bi-gram word/POS models.

label	number	average length
AMBI	138	1.69
CONF	88	2.36
COMP	88	2.94
SING	59	3.27
READ	243	1

Table 2: Error labels in the evaluation data set.

4.2 Evaluation data

To evaluate our proposed method and the model trained by automatically generated data, we build evaluation data. Our evaluation data are composed of 319 technical requirements (481 sentences with 10,324 tokens), extracted from 12 documents (over 200 pages) coming from four different companies, kept anonymous at their request. The main features considered to validate our data are as follows:

(1) requirements corresponding to various professional activities: product design, management, finance, and safety
(2) requirements following various kinds of business style and format guidelines imposed by companies
(3) requirements coming from various industrial areas: finance, telecommunications, transportation, energy, computer science.

To build evaluation data, we annotate POS labels using an HMM model and we correct them. Then, we manually assign error labels as defined in §3. Table 2 shows the number and the average length of error labels in the evaluation data set.

4.3 Results

Table 3 presents evaluation results based on the evaluation data that we described in §4.2. We also provide precision and recall for each error label. We obtain overall 79.17% F_1 score for our *automatic* error label annotation by using the CRF model learned from which we build training data.

4.4 Error analysis and discussion

READ error labels are entirely based on lexical information and we correctly annotate almost all of them because we have enough lexical information in training data. CONF error labels show only about 26% of precision because even though the expected modals

label	precision	recall	F_1
AMBI	45.65	82.89	58.88
CONF	26.14	82.14	39.66
COMP	73.86	84.42	78.79
SING	79.66	65.28	71.76
READ	100.00	99.18	99.59
total	71.59	88.55	**79.17**

Table 3: Evaluation results: We learned our model from the training data described in §4.1 and evaluated it using CRFs.

as erroneous should have been detected exclusively in the main clauses, many of them were identified in the subordinated clauses where their use is allowed. AMBI and SING error labels for the combinator error pattern are required parsing results, and we used them for building training data. However, our CRF model uses only lexical and POS information and it was difficult to spot the correct range of arguments of the combinators without syntactic information. Actually, by using our program that we introduced for building training data, which also used syntactic analysis, we can obtain up to 82.47%. Results on AMBI and SING labels for combinators error patterns by our heuristic rule-based program are especially better than by the CRF model. Note that our heuristic rule-based program are overfitted to our training/evaluation data and the proposed CRF model is better for the general usage of the error annotation. However, dependency information is difficult to be integrated in the CRF model with dependency distance. Moreover, dependency results are not often correct for conjunction marks such as *or* and *and*, which we use for the combinator error pattern.

5 Conclusion and Future Perspectives

We have presented a linguistic model for the requirements quality analysis. A tool helping to improve the requirements authoring quality allows to reduce multiple proofreading steps which are time consuming and costly but crucial in the whole life cycle of the Requirement Engineering. The accuracy of this kind of tools is obviously very important as technical authors (users) can reject to use them once they generate false positives of more than 20%. To reduce the rate of false positives, the model that we developed is based on the error patterns manually identified in the linguistic framework of the requirements grammar.

The results of our first experiments on the model developed in this paper show promising improvements and some directions for the future work. First, we need to improve the results of CONF and AMBI error patterns by increasing the accuracy of the dependency parsing results. For CONF, we can limit the identification of modal *shall* followed by an infinitive verb to those only preceded by the subject of the main clause. For AMBI, We are planning to effectively integrate syntactic analysis results in our learning model. Second, we can enrich the error patterns depending on the lexical information by adding more lexical items into our model. Third, the five constraints and the corresponding rules presented in the requirements grammar do not cover all of the potential errors of the requirements authoring. It is necessary to revise and complete the rules of the requirements grammar in order to detect another error types: (1) detection of incomplete terms (use of ”TBD”, ”TBC”, ”etc.”...) for the Completeness, (2) detection of over-specified elements (design/solution parts (how the system realize the required action) included in the requirements) for the Singularity, (3) detection of grammatical errors (e.g. ditransitive verbs missing one of arguments like *the system shall send the received configuration*) for the Completeness. There are also another types of constraints more ambitious such as the problem of consistency and of redundancy between requirements or sets of the requirements. For those errors, we need to consider contextual information over a requirement sentence and to understand semantic meaning of the requirements and the relation between them. Finally, the current model are based on automatically annotated training data. We can improve its quality by adding another language processing models to get the better result on error filtering method. We may also add eventually manual verification and we leave them as our future work.

References

[Alred et al.2012] Gerald J. Alred, Charles T. Brusaw, and Walter E. Oliu. 2012. *The Handbook of Technical*

Writing. Bedford/St. Martin's, New York.

[Arora et al.2013] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. 2013. Automatic Checking of Conformance to Requirement Boilerplates via Text Chunking: An Industrial Case Study. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 35–44, oct.

[Baroni et al.2009] Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. 2009. The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226.

[Berrocal Rojas and Barrantes Sliesarieva2010] Allan Berrocal Rojas and Elena Gabriela Barrantes Sliesarieva. 2010. Automated Detection of Language Issues Affecting Accuracy, Ambiguity and Verifiability in Software Requirements Written in Natural Language. In *Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas*, pages 100–108, Los Angeles, California. Association for Computational Linguistics.

[Berry et al.2003] Daniel M. Berry, Erik Kamsties, and Michael M. Krieger. 2003. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity.

[Bourque et al.2004] Pierre Bourque, Alain Abran, Juan Garbajosa, Gargi Keeni, Beijun Shen, Alain April, Antonia Bertolino, Durba Biswas, Nabendu Chaki, Roger Champagne, Christof Ebert, Pierce Gibbs, Mira Kajko-Mattsson, Gerald Kotonya, Eda Marchetti, James McDonald, Xin Peng, Annette Reilly, Pete Sawyer, Michael Siok, Yanchun Sun, and Hengming Zou. 2004. *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*. IEEE Computer Society.

[Brants2000] Thorsten Brants. 2000. TnT – A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231, Seattle, Washington, USA. Association for Computational Linguistics.

[Fabbrini et al.2001] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. 2001. An Automatic Quality Evaluation for Natural Language Requirements. In *in Proceedings of the Seventh International Workshop on RE: Foundation for Software Quality (REFSQ'2001*, pages 4–5, Interlaken, Switzerland.

[Firesmith2003] Donald Firesmith. 2003. Specifying Good Requirements. *Journal of Object Technology*, 2:77–87.

[Fuchs2012] Norbert E. Fuchs, 2012. *First-Order Reasoning for Attempto Controlled English*, pages 73–94. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Galin2003] Daniel Galin. 2003. *Software Quality Assurance: From Theory to Implementation*. Pearson.

[Glas2002] Robert L. Glas. 2002. *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional.

[Gnesi et al.2005] Stefania Gnesi, Fabrizio Fabbrini, Mario Fusani, and Gianluca Trentanni. 2005. An automatic tool for the analysis of natural language requirements. *CRL Publishing: Leicester*, 20:53–62.

[Grady2013] Jeffrey O. Grady. 2013. *System Requirements Analysis*. Elsevier.

[Hull et al.2011] Elizabeth Hull, Ken Jackson, and Jeremy Dick. 2011. *Requirements Engineering*. Springer-Verlag London.

[Kuhn2013] Tobias Kuhn. 2013. A Principled Approach to Grammars for Controlled Natural Languages and Predictive Editors. *Journal of Logic, Language and Information*, 22(1):33–70.

[Lavergne et al.2010] Thomas Lavergne, Olivier Cappé, and François Yvon. 2010. Practical Very Large Scale CRFs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 504–513, Uppsala, Sweden. Association for Computational Linguistics.

[Nivre et al.2006] Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*.

[Stecklein et al.2004] Jonette M. Stecklein, Jim Dabney, Brandon Dick, Bill Haskins, Randy Lovell, and Gregory Moroney. 2004. Error Cost Escalation Through the Project Life Cycle. In *Proceedings of the14th Annual International Symposium*, Toulouse, France.

[Weiss1990] Edmond H. Weiss. 1990. *100 Writing Remedies: Practical Exercises for Technical Writing*. Greenwood.

[White and Schwitter2009] Colin White and Rolf Schwitter. 2009. An Update on PENG Light. In Luiz Pizzato and Rolf Schwitter, editors, *Proceedings of the Australasian Language Technology Association Workshop*, pages 80–88, Sydney, Australia.

[Zapata Jaramillo2010] Carlos Mario Zapata Jaramillo. 2010. Computational Linguistics for helping Requirements Elicitation: a dream about Automated Software Development. In *Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas*, pages 117–124, Los Angeles, California. Association for Computational Linguistics.