

Problem of Incompleteness in Textual Requirements Specification

David Šenkýř^a and Petr Kroha^b

Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Keywords: Requirements Specification, Text Processing, Grammatical Inspection, Incompleteness, Domain Model.

Abstract: In this contribution, we investigate the incompleteness problem in textual requirements specifications. Incompleteness is a typical problem that arises when stakeholders (e.g., domain experts) hold some information for generally known, and they do not mention it to the analyst. A model based on the incomplete requirements suffers from missing objects, properties, or relationships as we show in an illustrating example. Our presented methods are based on grammatical inspection, semantic networks (*ConceptNet* and *BabelNet*), and pre-configured data from on-line dictionaries. Additionally, we show how a domain model has to be used to reveal some missing parts of it. Our experiments have shown that the precision of our methods is about 60–82 %.

1 INTRODUCTION

Textual requirements do not contain complete information about the system to be constructed. There are more reasons for that.

First, the software system describes only a specific part – a simplified model omitting some details – of the real system to be modelled. We are interested only in a subset of all existing objects, properties, and relationships the real system consists of. For example, when writing information systems, this subset is given by supposed queries. So, the first source of incompleteness is a natural one. We do not model and describe the complete reality, but only the part of it that we need to answer the user's queries.

Second, stakeholders working with the analyst on the textual version of requirements suppose that some facts are obvious, and they do not mention them. However, what is obvious for a user of a biological information system, is usually not obvious for a computer scientist.

Third, some details are forgotten at the very beginning, or some queries are appended later without to worry whether the underlying model contains the necessary objects, properties, or relationships.

Using incomplete information in requirements leads to incomplete or bad models. The resulting program has to be laboriously enhanced. In (Femmer et al., 2017), there is incompleteness together with

ambiguity and other nature language processing defects called as *requirements smells*, similarly to the concept of *code smells*.


In this paper, we investigate possibilities of how to reduce incompleteness in textual requirements using domain models and knowledge bases. However, the semantic relations in real systems are often so complex and hidden. Thus, they cannot be solved automatically. We generate a warning message in the case when we find a suspicious formulation.


In our paper, we focus on the following research questions.

1. Is it possible to indicate that some information has been omitted or forgotten in the text of requirements specification?
2. How to find sentences and their parts that are probably not complete?

We discuss these research questions in detail in Section 4 and answer them in Conclusion.

Our paper is structured as follows. In Section 2, we discuss related works. We present the problems of incompleteness in Section 3. Our approach is presented in Section 4, and we comment it on the illustrating example in Section 5. The implementation, used data, experiment, and results are described in Sections 6 and 7. Finally, in Section 8, we conclude.

^a  <https://orcid.org/0000-0002-7522-3722>

^b  <https://orcid.org/0000-0002-1658-3736>

2 RELATED WORKS

In this section, we discuss some key areas and approaches related to the research of software-assisted detection of incompleteness in textual requirements.

2.1 Incompleteness Detection Tools

Tool called *Cordula* (Compensation of Requirements Descriptions Using Linguistic Analysis), presented in (Bäumer et al., 2019), follows the vision of *On-the-Fly (OTF) Computing*. This vision assumes ad hoc providing software services based on requirements description written by a user in natural language. The authors of the paper discuss the question “How users might be involved in the compensation process?”. *Cordula* represents a chat bot (Friesen et al., 2018). The chat bot can highlight the inaccuracies, and a user can confirm the proposed suggestion by the chat bot or reject it. The proposed approach engages the users in the following steps. First, they describe their requirements in a short text. Based on that, they receive a response by the chat bot. Second, they are asked to react to the proposed suggestion. Based on that, the chat bot can adapt to the situation.

In the prerequisite paper (Bäumer and Geierhos, 2018), there are presented methods of quality violations in a requirements specification. The authors call them *linguistic triggers*. Besides the problem of incompleteness, there is also presented an approach of ambiguity detection. In the case of incompleteness triggers, authors pursue already presented approach divided into two steps (Bäumer and Geierhos, 2016). First, the detection via *predicate argument analysis* in which a *semantic role labeler (SRL)* assigns semantic roles such as *agent*, *theme*, and *beneficiary* to the recognized predicate. Presented illustrating example is verb “send” that is a three-place predicate because it requires the agent (sender), the theme (sent) and the beneficiary argument (sent-to). If the beneficiary is not specified here, it is unknown whether one or more recipients are possible. The second step is compensation. The authors state that they gathered software descriptions and their corresponding reviews from site <https://download.cnet.com>. Using *similarity search component* known from *information retrieval (IR)* domain, they try to find the potentially missing part *sent-to*.

The authors of (Eckhardt et al., 2016) focused on *performance requirements*. The requirements that describe system behaviour. They created an UML model reflecting 3 categories of performance requirements – *time behaviour* requirements (e.g., “The operation must have an average response time of

less than 5 seconds.”), *throughput* requirements (e.g., “The system must have a processing speed of 100 requests/second.”), and *capacity* requirements (e.g., “The system must support at least 50 concurrent users.”). Following the UML model, they are able to map a textual requirement to parts of the model. Based on predefined mandatoriness of each single part, they can indicate the possible missing parts. The mapping of textual requirements is done via *sentence patterns* derived from the model.

Tool called *NLARE* (A Natural Language Processing Tool for Automatic Requirements) is presented in the papers (Huertas and Juárez-Ramírez, 2013) and (Huertas and Juárez-Ramírez, 2012). There is a functional requirement defined as a finite set of words where words can form one of 3 three elements: *actor* (performs the action in the statement), *function* (indicates what action needs to be performed), or *detail* (indicates conditions under what action be expected). Based on this categorization, the incompleteness exists if any of the mentioned element is not found. Mapping of words and elements uses patterns represented by regular expressions that operate on part-of-speech tags.

2.2 Incompleteness Confrontation

The research in the corresponding area of processing of textual corpora also takes into the problem of incompleteness. For example, in (Rodriguez et al., 2018), authors propose methods of knowledge acquisition from Wikipedia, and they argue that it should be a source of minimization of incompleteness.

Paper (Femmer et al., 2017) classifies the problem of incompleteness as one of the *requirements smells*. Following ISO 29148 requirements engineering standard, they introduce a category of *requirements smell* called *incomplete references* that should be processed in their tool *Smella*. The detection mechanism is not in the scope of their paper.

Also paper with a concise name “What Did You Mean” (Geierhos et al., 2015) refers to the problem of incompleteness. They follow (Fabbrini et al., 2001) where the incompleteness is indicated by *under-specification*. An under-specification indicator is pointed out in a sentence when the subject of the sentence contains a word identifying a class of objects without a modifier specifying an instance of this class. Concerning the example sentence – “The system shall be able to run also in case of attack.” – we ask which attack the writer means?

3.2.1.19 Functional Requirement 1.19

ID: FR19

TITLE: Mobile application – Sorting results

DESC: When viewing the results in a list, a user should be able to sort the results according to price, distance, restaurant type, specific dish or restaurant name.

- When sorting by restaurant name, specific dish or restaurant type the results should be ordered alphabetically.
- When sorting by price the results should be ordered from cheapest to most expensive.
- When sorting by distance the results should be ordered from closets to furthest distance according to the user's position.

How is the distance measured?
 I.e., in km of air-line distance,
 in km of walk distance or in minutes of walk distance,
 using subway is included or not, etc.

A list of what?

A price of what?
 I.e., price of drinks,
 price of vegetarian meals,
 price of non-vegetarian meals, etc.

A restaurant can have more attributes.

What does it mean exactly?

The type can describe:
 geographical origin (chinese)
 or type of dishes (vegetarian)
 or type of service (take away)
 or quality of service (fast food)
 or some constraints (BYO – bring your own, no smoking).

Figure 1: Example of Incompleteness (Restaurant).

2.3 Related Works – Overview

The presented papers use, similarly to us, methods of nature language processing in the sense of resolving *part-of-speech tags* and *dependency structure* of a sentence. Our work differs in the usage of support resources. In Section 4, we present the sources that we use – prepared static collection of collocations and on-line resources *ConceptNet* and *BabelNet*. Our implemented tool analyzes the whole document at once and generates warnings. This is the difference compared with the mentioned chat bot in (Friesen et al., 2018).

3. ...distance → how is the distance measured, i.e., in km of air-line distance, in km of walk distance or in minutes of walk distance using subway included?
4. ...restaurant type → what does it mean exactly? The type can describe, e.g., a geographical origin (Chinese), a type of dishes (vegetarian), a type of service (take away), a quality of service (fast food), some other constraint (BYO = bring your own, no-smoking, etc.).

In Example 1, the source of incompleteness is that some attributes of the class *Restaurant* are not clearly and uniquely defined.

3 PROBLEM OF INCOMPLETENESS

To illustrate the problem of incompleteness, we introduce two typical examples.

(Example 1)

In description of *functional requirement Nr. 1.19* in Fig. 1 taken from (Geagea et al., 2010), we can see some incomplete formulations. The user should have the possibility to sort the list of found restaurants according to attributes that are not uniquely defined. We extracted the following sources of incompleteness:

1. ... viewing the results as a list → a list of what?
2. ... a user should be able to sort the results according to price → a price of what? A price of drinks, a price of vegetarian meals, a price of non-vegetarian meals or a price of something else?

(Example 2)

Suppose, we want to write requirements specifications for university information systems. Modelling attributes of classes *Student* and *Teacher*, we need to include attributes like *First Name*, *Last Name*, *Address*, etc.

However, we do not include the attribute *Shoe-Size* in the case of University of Technology, but we have to include it in the case of Military University because of a uniform. This means that the completeness of the set of attributes of class *Student* can be decided only with respect to the set of queries or operations that concern the data stored.

In Example 2, the set of queries used by the information system of a Military University contains the attribute *Shoe-Size*, but the set of queries used by University of Technology does not.

4 OUR APPROACH TO PROBLEMS OF INCOMPLETENESS

The goal of our project is to identify sources of incompleteness in textual requirements. We compare their content with:

- the usual usage of words – group A,
- the semantic knowledge stored in our model – group B,
- the information contained in the set of proposed queries as sources of actions to be performed – group C,
- the draft of the pre-generated model – group D.

When we find some discrepancy, we generate a warning message that signalises the necessity of human intervention.

4.1 Group A (Usual Usage of Words)

Nouns in sentences are investigated using a specific, predefined set of words. If the kind of usage in the textual requirements (e.g., a list) does not correspond to the kind of usage in the vocabulary (e.g., a list of) then a warning message is generated.

The set of words we used for this purpose is available on our web-page¹, hereinafter referred to as a *common noun and preposition collocation set*. We build this collection based on various collocation web-pages. Each entry is provided with a resource reference. We would like to mention *Free Online Collocations Dictionary*² from *ProWritingAid*. The advantage of this dictionary is that you can find collocation also via preposition (e.g., *of*). Based on the query, the result contains collections of common used collocated words grouped by the *part-of-speech* category. Our second significant resource is *Corpus of Contemporary American English*³ (COCA). You can search the corpus via expression containing *part-of-speech* tags. In our situation, we were looking for common collocation, e.g., of preposition *of*. Therefore, we inserted “[nn*] of” query expression. The result list contains founded words sorted by the frequency of usage.

4.2 Group B (Semantic Knowledge)

The semantic knowledge stored in our model of the reality contains some specific information, e.g., the

¹<https://temos.ccmi.fit.cvut.cz/sources>

²<https://prowritingaid.com/en/Collocation/Dictionary>

³<https://www.english-corpora.org/coca>

information that not every restaurant has to have a beer in a drink menu. This kind of information corresponds with possible (but avoided) existence of adjectives before nouns. Our source of information is *ConceptNet*⁴ (Speer and Havasi, 2012), in which we can find that there exists the relation *Is-A* of name Types of Restaurants. Coming from this information, we generate a question concerning the possibility of the missed adjective before the word restaurant.

Our second source is *BabelNet*⁵ (Navigli and Ponzetto, 2012) that provides *Has Kind* relation. If some examples are found for a specific noun, we generate a question concerning the possibility of missed specialisation of the entity.

4.3 Group C (Actions)

Verbs in sentences of queries are investigated in the sense whether the action, that they describe, can be performed in the existing model (e.g., sorting without a key, sorting without a unique key, the number of shoes of size 42 for students of military university).

As a prerequisite, we are able to check relation in the way of correct usage of the verb. The English verbs can take 0, 1, or 2 objects, depending on the verb. Verbs without objects are called *intransitive*, and the other ones are called *transitive*. Using the dependencies recognition, we check if the verb has any objects. If no object is found, we check the verb against the list of intransitive verbs (e.g., Wiktionary collection of such verbs⁶). For example, the standalone sentence “A warning appeared.” does not bring new information, but it is grammatically correct. On the contrary, the standalone sentence “Administrator needs to maintain.” contains transitive verb *need*, and we are missing the information about what is need to be maintained. Therefore this sentence is suspicious and TEMOS generates warning for the user.

4.4 Group D (Model Validation)

We process each sentence one by one and incrementally build the corresponding UML class model (Šenkýř and Kroha, 2018).

When the draft of our model is ready, we have a chance to check following simple indicators of missing or unrecognized information:

- “empty” class without attributes,
- class with *no relation* to any other class.

⁴<http://conceptnet.io>

⁵<http://live.babelnet.org>

⁶https://en.wiktionary.org/wiki/Category:English_intransitive_verbs

5 ILLUSTRATING EXAMPLE

We use the textual requirements specification to build the corresponding UML model in the first step, as we described in (Šenkýř and Kroha, 2018).

To introduce the problem of *incompleteness*, we continue in the analysis of Example 1 from the previous section. Our tool TEMOS delivers the first version of classes, attributes, and relationships including the class representing restaurant: `class RESTAURANT(Name, Address, Capacity, Open Hours, Menu, Drink menu, Parking, etc.)`

The procedure of analysing the text of *DESC* in *FR19* will run as follows:

1. Oxford Advanced Learner's Dictionary⁷ defines "list" noun as *a series OF names, items, figures, etc.* According to the definition, this means that the "... OF WHAT" part may have been omitted (Group A, Section 4.1).
 - Our tool TEMOS generates the first warning message: "*FR 19 – DESC: A list OF WHAT should be generated?*"
 - After a discussion with stakeholders, the analyst writes a new formulation: "*DESC: When viewing the results in a list OF RESTAURANTS, a user should be able to sort the RESTAURANTS according to price, ...*"
2. As the first sentence analysis continues – "...to sort the restaurants according to price" (assumption: existence of a unique sort key among attributes of objects to be sorted, i.e., of the corresponding class), our tool TEMOS checks whether the class *Restaurant* has an attribute *Price* that can be used for sorting – information of the Group B (4.2). The attribute *Price* is among attributes of the class *Menu* and among attributes of the class *Drink menu*, but it is not among attributes of the class *Restaurant*.
 - As it is not the case, TEMOS generates the following warning message: "*Restaurants as results of the search cannot be sorted according to the attribute Price, because Price is not an attribute of the class Restaurant. The class Restaurant has the following attributes: Name, Address, Capacity, Open Hours, Menu, Drink menu, Parking ...*"
 - Instead of that, the class *Restaurant* has an attribute *Menu* containing names of dishes and beverages, including their prices in some form.

⁷https://www.oxfordlearnersdictionaries.com/definition/english/list_1

So, we can generate a warning message and, after a discussion with stakeholders, we can use, e.g., the price of the cheapest beer from the restaurant drink menu as a sorting key. After a discussion with stakeholders, the analyst writes a new formulation:

"DESC: When viewing the results in a list of restaurants, a user should be able to sort the restaurants according to the price of the cheapest beer in the drink menu of each restaurant."

This formulation assumes that there is a unique cheapest beer in a drink menu of every restaurant. However, it may be a wrong assumption.

3. Our tool TEMOS generates the following warning message: "*There is neither a constraint assigned to the class Restaurant that each restaurant has to serve beer, nor a constraint that the drink menu contains a unique cheapest beer. So, the formulation is incomplete because the sorting key has to exist and it has to be unique.*"

After a discussion with stakeholders, the analyst writes a new formulation:

"DESC: When viewing the results in a list of restaurants, a user should be able to sort the restaurants according to the price of the cheapest drink in the drink menu of each restaurant."

The situation is even more complicated if we should sort restaurants according to their distance from the user's position. Without sorting, it would be enough to show a map to the user with his/her position and with the position of the restaurant. Unfortunately, this cannot be used for sorting.

If we were interested in air-line distance, we would need the GPS-coordinates of the restaurants would belong to attributes of the class *Restaurant*, and we need to obtain the GPS-coordinates of the user's position. However, the air-line distance is not very useful in many towns. Often, it is much more important to know how much time we need to achieve the goal place. So, we speak about a distance, but we mean the time interval. Other metrics, e.g., Manhattan metric, using a subway, or a taxi route, are complicated too. As we can see, a simple formulation of requirements can cause implementation problems. The description in details is out of the scope of this paper.

6 IMPLEMENTATION

In (Bäumer et al., 2019), tools are distinguished in two categories:

- tools focused on specific linguistic inaccuracy,

Table 1: Frequency of Generated Warnings.

Software Requirements Specification	WC	WA	FP	FNLP	PRE
E-Voting Systems (Daimi et al., 2006)	1243	26	8	3	69.23
Restaurant Menu & Ordering System (Henning et al., 2008)	1266	25	10	2	60.00
Amazing Lunch Indicator (Geagea et al., 2010)	3606	64	18	5	71.88
Online National Election Voting (Joldoshev et al., 2010)	3726	55	10	5	81.81

Legend: **WC** – word count, **WA** – number of warnings, **FP** – number of false positive warnings from **WA** column, **FNLP** – number of false positive warnings from **FP** column due to bad NLP categorization, **PRE** – precision (%)

- tools capable of identifying multiple linguistic inaccuracies.

Based on this categorization, our *TEMOS* (Textual Modelling System) tool belong to the second category. So far, we are able to generate the UML class diagram (with a simple model validation) (Šenkýř and Kroha, 2018) and indicates ambiguity issues (Šenkýř and Kroha, 2019) and incompleteness issues (presented in this paper). In this case of incompleteness detection, the input is a plain text. As an output, our *TEMOS* tool generates warning messages.

The current version of our tool is written in Python and it is powered by *spaCy*⁸ NLP framework. Following text document analysis pipeline presented in (Šenkýř and Kroha, 2018), for incompleteness resolving, we need preprocessing in the form of *tokenization*, *sentence segmentation*, *part-of-speech tagging*, *lemmatization*, and *dependencies recognition*. We reuse the already presented idea of a *grammatical inspection* and *sentence patterns* following (Rolland and Proix, 1992). We select the first group to present a details of patterns usage.

6.1 Patterns of Group A

The search for nouns belonging to group A (Section 4.1) can be done using a simple sentence pattern presented in Fig. 2 that uses corresponding part-of-speech tags and *preposition* dependency relation.

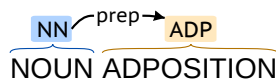


Figure 2: Noun with Preposition Pattern.

If this pattern is matched, then no further action is needed. Otherwise, we will look at the mentioned preconfigured *common noun and preposition collocation set* to check if the tested noun is to be found together with a preposition or not.

To precise this method, in the case when the simple pattern is not matched, we use the white-list of supporting patterns. The purpose of these patterns is

that, during the testing phase, we indicate some repetitive parts of sentences that should be not indicated as a warning. For example, we can show the white-list pattern of common sentence start illustrated in Fig. 3. When some of the white-list patterns is matched, no warning is generated, and the analysis continues with the next word.

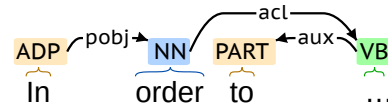


Figure 3: “In Order to Action” Pattern.

7 DATA, EXPERIMENT, RESULTS

We tested our *TEMOS* tool on the selection of four freely available specifications from the Internet (Daimi et al., 2006), (Henning et al., 2008), (Geagea et al., 2010), (Joldoshev et al., 2010). From every document, we take only the text parts representing requirements. In this experiment, we did not distinguish between functional and non-functional requirements.

For each generated warning, we tried to decide whether it could contribute to refining the requirements or not (and it is therefore false positive). Above, in Table 1, we present the results of *Group A analysis* in the form of the number of generated warnings by our tool together with the information about the number of false positive warnings. In some cases, the warning was generated because of bad NLP categorization of *part-of-speech tag* or *dependency tag*. We also stated this information in Table 1. The documents are ordered by the number of words devoted to requirements.

The result of this experiment shows that for the selected documents, the benefit of the grammatical inspection analysis ranges between 60–82 % of reasonable warnings. To be fair, it is needed to state that documents also contain a glossary of used terms and some of them also diagrams that can help clarify some of the generated warnings. On the other hand, not all stakeholders have to understand the diagram notation.

⁸<https://spacy.io>

8 CONCLUSIONS

Because of the phenomena of inaccuracies in the textual requirements specifications, we follow our previous work and, in this contribution, we focus on the problem of incompleteness. In Section 3, we presented illustrating examples of incomplete requirements and in Section 4, we discussed techniques how to warn users about a potential problem.

Based on that, we extended our TEMOS tool with the methods of grammatical inspection dedicated to checking the usage of *nouns* (representing future information system entities) and *verbs* (representing future relations and actions) together with the simple methods dedicated to checking the incompleteness of background generated UML class model in the form of entities without attributes and entities with no relation. Therefore, we are able to check incompleteness on the level of single requirement and on the level of incomplete requirements specification as a whole. Some of the proposed methods use *ConceptNet* semantic network, *BabelNet* semantic network, and pre-configured data from on-line dictionaries. Following the presented experiment, we can conclude that also methods of grammatical inspection can generate reasonable warnings for the users.

Concerning our research questions, we mentioned in Introduction in Section 1; there are the following answers.

1. We argue that some incompleteness symptoms can be indicated.
2. We developed new suitable methods, see Section 4, that generate warnings to the analyst. He or she has to decide, in cooperation with stakeholders, how to complete the text of requirements specification.

In our further research, we extend the possibilities of the textual requirements specification checking by investigating formulations that concern the dynamic UML model. The next problem, we have to solve is traceability. We would like to have a clear overview, which parts of the model will change after a specific part of the textual requirements specification will be modified.

So far, we have been dealing with UML class model generation and ambiguity and incompleteness indication. Another typical issue of textual requirements is also the problem of inconsistency. This topic is the subject of our further research, too.

ACKNOWLEDGEMENTS

This research was supported by the grant of Czech Technical University in Prague No. SGS17/211/OHK3/3T/18.

REFERENCES

- Bäumer, F. S. and Geierhos, M. (2016). Running Out of Words: How Similar User Stories Can Help to Elaborate Individual Natural Language Requirement Descriptions. In Dregvaite, G. and Damasevicius, R., editors, *Information and Software Technologies*, volume 639, pages 549–558. Springer International Publishing, Cham.
- Bäumer, F. S. and Geierhos, M. (2018). Flexible Ambiguity Resolution and Incompleteness Detection in Requirements Descriptions via an Indicator-Based Configuration of Text Analysis Pipelines. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, pages 5746–5755.
- Bäumer, F. S., Kersting, J., and Geierhos, M. (2019). Natural Language Processing in OTF Computing: Challenges and the Need for Interactive Approaches. *Computers*, 8(1):14.
- Daimi, K., Snyder, K., James, R., and Park, A. (2006). Requirements Engineering for E-Voting Systems. Available from: <https://pdfs.semanticscholar.org/318f/989bc774c9c3e907c470ebb3b1016672f679.pdf>.
- Eckhardt, J., Vogelsang, A., Femmer, H., and Mager, P. (2016). Challenging Incompleteness of Performance Requirements by Sentence Patterns. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 46–55, Beijing, China. IEEE.
- Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. (2001). An Automatic Quality Evaluation for Natural Language Requirements. In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 1.
- Femmer, H., Méndez Fernández, D., Wagner, S., and Eder, S. (2017). Rapid Quality Assurance with Requirements Smells. *Journal of Systems and Software*, 123:190–213.
- Friesen, E., Bäumer, F. S., and Geierhos, M. (2018). COR-DULA: Software Requirements Extraction Utilizing Chatbot as Communication Interface. In Schmid, K., Spoletini, P., Ben Charrada, E., Chisik, Y., Dalpiaz, F., Ferrari, A., Forbrig, P., Franch, X., Kirikova, M., Madhavji, N., and et al.Editors, editors, *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018)*. CEUR-WS.org.
- Geagea, S., Zhang, S., Sahlin, N., Hasibi, F., Hameed, F., Rafiyan, E., and Ekberg, M. (2010). Software requirements specification: Amazing lunch indicator. Available from:

- http://www.cse.chalmers.se/~feldt/courses/reqeng/examples/srs_example_2010_group2.pdf.
- Geierhos, M., Schulze, S., and Simon Bäumer, F. (2015). What Did You Mean? Facing the Challenges of User-generated Software Requirements. In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 277–283, Lisbon, Portugal. SCITEPRESS – Science and Technology Publications.
- Henning, T., Keehn, D., Thompson, J., and Wildermoth, M. (2008). Software Requirements Specification: Restaurant Menu & Ordering System. Available from: <https://kungfumas.files.wordpress.com/2017/09/099.pdf>.
- Huertas, C. and Juárez-Ramírez, R. (2012). NLARE, A Natural Language Processing Tool for Automatic Requirements Evaluation. In *Proceedings of the CUBE International Information Technology Conference on - CUBE '12*, pages 371–378, Pune, India. ACM Press.
- Huertas, C. and Juárez-Ramírez, R. (2013). Towards Assessing the Quality of Functional Requirements Using English/Spanish Controlled Languages and Context Free Grammar. In *The Third International Conference on Digital Information and Communication Technology and its Applications (DICTAP2013)*, pages 234–241.
- Joldoshev, E., Matar, H. S., Özkan, M. B., and Lutin, H. (2010). Software Requirements Specification for Online National Election Voting. Available from: <https://senior.ceng.metu.edu.tr/2011/iteam4/documents/srs-iTeam4.pdf>.
- Navigli, R. and Ponzetto, S. P. (2012). BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250.
- Rodriguez, D. V., Carver, D. L., and Mahmoud, A. (2018). An Efficient Wikipedia-Based Approach for Better Understanding of Natural Language Text Related to User Requirements. In *2018 IEEE Aerospace Conference*, pages 1–16, Big Sky, MT. IEEE.
- Rolland, C. and Proix, C. (1992). A Natural Language Approach for Requirements Engineering. In *Advanced Information Systems Engineering*, pages 257–277, Berlin, Heidelberg. Springer.
- Šenkýř, D. and Kroha, P. (2018). Patterns in Textual Requirements Specification. In *Proceedings of the 13th International Conference on Software Technologies*, pages 197–204, Porto, Portugal. SCITEPRESS – Science and Technology Publications.
- Šenkýř, D. and Kroha, P. (2019). Patterns of Ambiguity in Textual Requirements Specification. In Rocha, Á., Adeli, H., Reis, L. P., and Costanzo, S., editors, *New Knowledge in Information Systems and Technologies*, volume 1, pages 886–895, Cham. Springer International Publishing.
- Speer, R. and Havasi, C. (2012). Representing General Relational Knowledge in ConceptNet 5. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages

3679–3686, Istanbul, Turkey. European Language Resources Association (ELRA).