Problem of Inconsistency in Textual Requirements Specification

David Šenkýř[a] and Petr Kroha[b]
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Keywords: Textual Requirements Specification, Text Processing, Inconsistency, Patterns, Domain Model.

Abstract: In this contribution, we investigate the inconsistency problem in the textual description of functional requirements specifications. In the past, the inconsistency problem was investigated using analysis of the UML models. We argue that some sources of inconsistency can be revealed in the very first steps of textual requirements analysis using linguistic patterns that we developed. We cluster the sentences according to their semantic similarity given by their lexical content and syntactic structure. Our contribution focus on revealing linguistic contradictions (e.g., a combination of passive voice, antonyms, negated synonyms, etc.) of facts and rules described in different parts of requirements together with contradictions of the internally generated model.

1 INTRODUCTION

Requirements specifications, their elicitation and analysis belong to the very first phase of the software development, and it is well-known that mistakes done in this phase are the most expensive ones. Using inconsistent information in requirements leads to the wrong model. After implementation and testing, the resulting program has to be later laboriously enhanced.

Textual requirements do not always contain consistent information about the system to be constructed. The reason is that they might be written by independent groups of stakeholders who have different interests, goals, backgrounds, and/or (incomplete) knowledge of the domain.

The conflicts may erase in the case when more than one requirement, i.e., a set of requirements *SR*, refers to the same object of the domain. It will be denoted as an overlap between requirements (Spanoudakis and Finkelstein, 1998). In this paper, we identify two kinds of *the core sources of the inconsistency* of textual requirements. First, semantic overlaps of requirements sentences, e.g., at least two contradicting assertions exist that concern the same subject and the corresponding verb with the object. Second, the incompleteness of requirements (we discussed it in our paper (Šenkýř and Kroha, 2019b)). Our goal is to reveal at least some of the inconsistencies, and to generate questions or warning messages that have to be answered by domain experts and analysts. The answers have to be used to edit the text of requirements. The text of the requirements is the only source of information. We divided the text analysis into two phases. The first phase builds a UML model from sentences using the grammatical inspection (Šenkýř and Kroha, 2018) – see Example 1 in Section 3.2. The second phase provides an analysis of sentences using our linguistic patterns with the goal to find contradictions in them. Part of it is the using information already stored in the UML model. We use the *part-of-speech* and *dependency analysis* of sentences to construct the corresponding oriented graphs, and we analyze sentences that describe the same event. We present our patterns that may indicate candidates for inconsistency.

Our paper is structured as follows. In Section 2, we discuss related work. In Section 3, we describe sources of inconsistency that we are trying to identify. Our approach is presented in Section 4. Our implementation is described in Section 5. Used data, experiments, and results are discussed in Section 6. In Section 7, we summarize the achieved results and conclude.

2 RELATED WORK

The problem of inconsistency of requirements specification has been investigated since the late '80s

[a] https://orcid.org/0000-0002-7522-3722
[b] https://orcid.org/0000-0002-1658-3736

Šenkýř, D. and Kroha, P.
Problem of Inconsistency in Textual Requirements Specification.
DOI: 10.5220/0010421602130220
In *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2021)*, pages 213-220
ISBN: 978-989-758-508-1

(Hunter and Nuseibeh, 1998). As basic papers and surveys, we can denote (Spanoudakis and Zisman, 2001) and (Kamalrudin and Sidek, 2015).

Some works are based on semi-formal specifications in the form of UML diagrams (Torre et al., 2018) and use methods of formal specifications (Schamai et al., 2016) to check the consistency. However, the adoption of formal methods is still not widely accepted by industries (Fanmuy et al., 2012). Other works use ontologies (Kroha et al., 2009) or OCL (da Silva and Fernandes, 2018).

Our approach uses informal specifications written in natural language. There is some similarity to the approach described in (Gargiulo et al., 2015). The difference is that in (Gargiulo et al., 2015), the authors use only RDF triplets representation of the textual requirements specifications, whereas we use the complete text. The knowledge represented in RDF consists of triplets ⟨subject, predicate, object⟩. However, it reduces the semantic meaning of most sentences drastically because of the omitted information carried by omitted words in sentences. We use structure similar to RDF triplets only to cluster sentences.

Computational linguistics is engaged in a problem of text entailment (in query answering). The edit distance of two sentences (text and hypothesis) is used as a measure. A similar approach to the contradiction of two sentences is used in (de Marneffe et al., 2008) – see Section 3.1.1. The difference to our approach is that linguists do not use the information stored in the UML model (Šimko et al., 2013) – see Example 1 in Section 3.2 to check the completeness that can be a source of inconsistency.

3 SOURCES OF INCONSISTENCY

In our investigation, we distinguish the following sources of inconsistency: inconsistency caused by semantic overlaps of sentences (Section 3.1), inconsistency caused by incompleteness (Section 3.2), and inconsistency caused by external information. Because of the limited scope of this paper, we do not discuss the last case here. However, we analyze it in our paper about default consistency rules (Šenkýř and Kroha, 2021).

3.1 Semantic Overlaps as Sources of Inconsistency

The idea of *inconsistency* starts from the intuitive concept of contradiction, which means that *a statement* and its *negation* are found to hold simultaneously (Gargiulo et al., 2015). A semantic overlap between two sentences occurs if these two sentences express their statements about the same situation. The problem is how to reveal the contradiction of statements because they are often expressed in some "camouflaged" forms using synonyms, antonyms, and others "linguistic tricks" (Section 3.1.1).

3.1.1 The Linguistic Sources of the Inconsistency

In (de Marneffe et al., 2008), there is a list of *contradiction types* from a linguistic point of view. We adapted it for purposes of our patterns as follows:

- using *antonyms*,
- using *a negation*,
- using a combination of *synonyms* together with *changed roles of subject and object*, *passive voice*, and *negation*, e.g., *"the user can edit a document"* contra *"the user cannot correct a document in this mode"* (here, we suppose that the verbs *"to edit"* and *"to correct"* have the same meaning) contra *"a document cannot be corrected by the user"*,
- using *numerically different data*, e.g., *"you will start the function by double click"* contra *"you will start the function by one click"*,
- using *factive contradiction* in the sense of attributes of the subject,
- using *lexical contradiction*, e.g., *"to obtain results stay joined and wait"* contra *"to obtain results restart the application"* contra *"to obtain results restart the system"*,
- using *world knowledge* to indicate the contradiction, e.g., *"there is public access to your private data"*.

Additionally, some words may change, influence, or limit the sense of the sentence, e.g., *but*, *except*, *however*, *instead of*, *when*, *so that*, *that*.

3.2 Inconsistency between the Text and the UML Model

Using the grammatical inspection in the first phase of our process, we construct a skeleton of the UML model, e.g., classes, attributes and their values, methods. When we later compare the sentences, in which the attribute and its values are mentioned, we test whether all attribute values are mentioned in formulations defining decisions about the object's behavior. If some of them are not mentioned, then this incompleteness can cause inconsistency, because it may happen that the behavior is not defined for the missing attribute values. Further, we test whether the chains of applied methods are the same in sentences of the similar semantics – see Example 2. In this way, we can find conflicts among descriptions of instances, classes, and relations of the static model, and also among descriptions of sequence diagram and state diagram of the dynamic model. We search for sentences that describe the same event and compare them. More details are given in Section 4.1.

In general, we suppose that the textual description of requirements specification contains:

- The sentence S_1, in which an attribute value of an object O is expected to be V_1 to start a specific action A in a specific context *Cont*.
- The sentence S_2, in which an attribute value of an object O is expected to be V_2 to start the same specific action A in a specific context *Cont*.

Example 1: Attributes of A Button.

Maybe, the following sentences are a part of a requirements specification:
1. *To exit the application, the user has to press the red button.*
2. *To exit the application, the user has to press the square-shaped button.*
3. *To exit the application, the user has to press the EXIT-button.*
Analyzing these requirements, we can derive that the class *Button* has the Button-attribute *color*, the Button-attribute *shape*, and the Button-attribute *label*. This means, these sentences as part of the requirements are confusing, but they are not inconsistent because a red, square-shaped button with the label EXIT may exist, i.e., it is possible to construct an object of class *Button*, whose attributes *color*, *shape*, and *label* have the described values.

Our task is to generate a message saying that there is a suspicion of inconsistency. In any case, this is a not preferable style of writing textual requirements.

(End of Example)

Another case, we found, concerns the sequence of actions that is stored in a sequence diagram. If the chain of actions differs in different sentences that should have the same effect, we generate a warning message.

Example 2: A Missing Segment in A Chain of Actions.

Sentence 1: To edit a file, the user has to open the file, make changes, save the file (or save the file as), and close the file.

Sentence 2: To edit a file, the user has to open the file, make changes, and close the file.

(End of Example)

4 PATTERNS AS OUR APPROACH

Our motivation and goal are to identify *suspicious textual formulations* that could be the source of inconsistency.

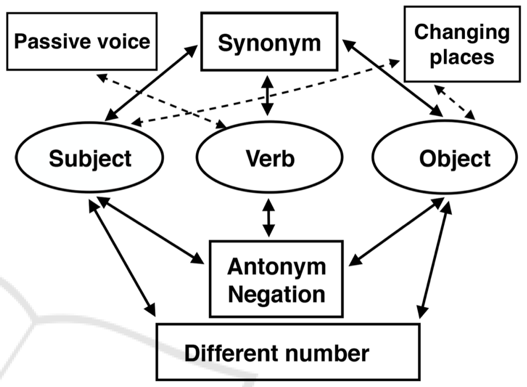

Figure 1: The idea of inconsistency patterns.

The idea of our approach is shown in Fig. 1. The basic structure is given by *subject–verb–object*. However, they can be expressed by related synonyms, antonyms, direct negation, and numerical data. So, the spectrum of possible descriptions of one statement is very rich. Our patterns should reveal it. Practical examples of data used for experiments are given in Section 6.1.

4.1 Model Construction and Semantically Similar Sentences

For our methods, the sentences containing the standard *couple of subject and verb* or the *to-infinitive clauses* are interesting. We benefit from mapping such sentences to *acts*. One reason is that one sentence can represent multiple acts. Our *act* is a tuple containing: the original *subject* of the sentence, the original *verb* of the sentence, the original *object* of the sentence, an indicator of whether the sentence contains *negation*, an *auxiliary verb* if present, an *adverbial modifier* (e.g., automatically) if present, a *predicate* (e.g., only one, each) if present, a *condition act* if present, *what* is the point of interest if recognized, *who* is responsible for the action if recognized.

To reduce the complexity of the analysis, we define semantically similar acts as follows:

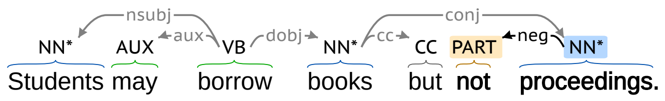

Figure 2: Matched Pattern #1 (Pure Negation).

- C_1: acts having the same subject, verb, and object,
- C_2: acts having the same subject and verb (passive mode),
- C_3: acts having the same subject (passive mode),
- C_4: acts having the same subject and object,
- C_5: acts having the same what-part and verb.

To understand the graphical representation of the presented *sentence patterns*, we explain it using the pattern in Fig. 3. We use *part-of-speech tags* provided by *spaCy*[1] (based on *Universal Dependencies v2 POS tag set*) where `VB` is a *verb*, `NN` is a *noun*, etc. Often, *subjects* and *objects* are represented by a composition of several nouns. Therefore, we introduce the shortened notation `NN*`, which means that at least one noun is required, but there should also be two or more nouns connected via *compound* relation. In the first phase, we need to consider the following steps:

- *negation recognition* – We check the pure negation of verbs (*can–can not–can't*) or compound nouns via Pattern in Fig. 3. In Fig. 2, there is this pattern matched as a part of matching standard *subject–verb–object(s)* pattern. Besides the *not* part, there are other words influencing the negation of meaning. Let's show the pattern of the word *except* in Fig. 4 as an example.

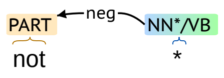

Figure 3: Pattern #1 (Pure Negation).

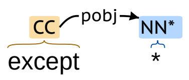

Figure 4: Pattern #2 (Except).

- *coreference recognition* – Clustering of sentences based on subject or object is challenging because of the *coreferences* of pronouns. This means that we are not simply looking for all sentences that have the same subject or object.
- *predicates recognition* – We identify parts of sentences that describe *predicates* – numeric ones (see Fig. 5) or determiner ones (see Fig. 6). The determiner should be represented by words like *each*, *every*, *all*, *any*, etc. We use the simple first-order logic definition saying that a predicate is a statement about the properties of an object that may be true or false depending on the values of its variables. Predicates can affect *actors of relations* as with relation *"borrow"* in example sentence in Fig. 7 or *aspects and limits of classes* via auxiliary verb *"be"* (see Fig. 8). Mapped predicates also may indicate *cardinalities* of relations between actors or objects. We find predicates concerning the same objects and the same attributes.

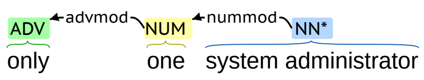

Figure 5: Pattern #3 (Numeric Predicate).

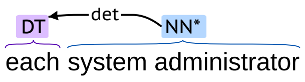

Figure 6: Pattern #4 (Determiner Predicate).

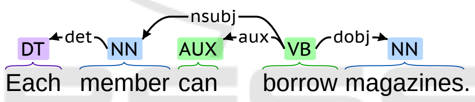

Figure 7: Relation actor predicate.

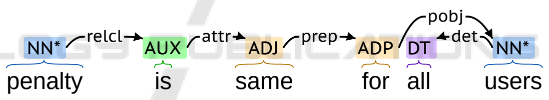

Figure 8: Pattern #5 (Predicate and Auxiliary Verb).

- *condition recognition* – Describing actions often includes a conditional part. The conditional parts of two sentences may have the same *subject–verb–object*. If one sentence considers the positive case and another sentence considers the negative case, then there would be a conflict generating a warning because of the negation of the action. We use these conditional parts as separate conditional acts (affecting standard acts) in the conflict resolution of the standard acts.

4.2 Example: Library Information System

We use and extend the simple example of *Library information system requirements* from (Spanoudakis and Finkelstein, 1998).

[1]https://spacy.io/api/annotation#pos-tagging

Example 3: Functional Requirements.

FR 1. *Users of the Library information system are students and staff members.*

FR 2. *Users borrow books.*

FR 3. *Holding an item is limited in time.*

FR 4. *Holding an item beyond the time limit carries a penalty, which is the same for all users.*

FR 5. *Students may borrow books but not proceedings.*

FR 6. *Staff members can borrow both books and proceedings.*

FR 7. *Students may borrow an item for up to 10 days. Holding an item beyond this period carries a penalty of 50 p per day.*

FR 8. *Staff members can borrow an item for up to 30 days. Holding an item beyond this period carries a penalty of 10 p per day.*

FR 9. *Each user except an administrator needs to change his/her password every three months.*

(End of Example)

The purpose of this example is to show the contradiction caused by teaching students. We use it to illustrate our approach. When using an algorithm to analyze these textual requirements instead of a human being analyst (like in (Spanoudakis and Finkelstein, 1998)), we can see the following problems:

1. In FR 1, there are two missing rules. In our paper (Šenkýř and Kroha, 2021), we call them *default consistency rules*. The first default consistency rule is *"All information systems need an administrator, who is a unique, specific user"*. The second default consistency rule is *"All libraries need library staff members, who are specific users of the library information systems, too"*.

So, the set of users is incomplete, and it has to be completed. In this case, we model the classes and the Is-a relation between *user* (as a superclass) and subclasses *student* and *university-staff-member*. Then, we generate a question asking about the completeness of this relation, i.e., whether the superclass *user* has only these two mentioned subclasses. The right answer contains not only the missing administrator and the missing library staff member but also the missing teaching students. After the subclasses *administrator*, *library staff member*, and *teaching student* will be included into the requirements, our system TEMOS will complain of incompleteness because the loan time periods and penalties are not defined for these three subclasses.

2. In FR 2, the formulation *"Users borrow books"* is misleading because users not only borrow books. They hold them and have to return them, too. Without this information, the time limit mentioned in FR 4 and the period mentioned in FR 7 and FR 8 makes no sense. Using linguistic analysis, we find that there are only the verbs *"borrow"* and *"change (the password)"*. Here, we can see the incompleteness as the source of the inconsistency again.

3. In FR 3, this formulation is understandable to a human being but not to an algorithm. *"Holding an item"* means *"holding a borrowed item"*. Our system generates a message: *"The requirements No. 3 is not understandable"*. The domain expert can answer: *"Users borrow books, hold them, and return them."*

4. In FR 5, there is matched *Pattern #1 (Pure Negation)* as shown in Fig. 2.

5. In FR 7 and FR 8, we can detect an inconsistency to FR 4. Either all library users pay the same penalty or different penalty according to their status.

6. In FR 9, there is *Pattern #4 (Determiner Predicate)* matched twice (*"each user"* and *"every three months"*) and *Pattern #2 (Except)* is matched once.

5 IMPLEMENTATION

Our tool *TEMOS* is a single-page web application. The current version of the back-end is written in Python, and it is powered by *spaCy*[2] NLP framework in version 2.2. We use texts written in English, and for this purpose, we use the pre-trained model called *en_core_web_lg* (available together with spaCy installation).

The workflow of our implemented tool extends the original one presented in (Šenkýř and Kroha, 2018). First, we use the base text classification provided by *spaCy*. It includes *tokenization*, *sentence segmentation*, *part-of-speech tagging*, *lemmatization*, *dependency recognition*, and *co-reference recognition*.

The next step is to check the text against inaccuracies. This check is optional but recommended before our tool starts to generate the output model. So far, we tackle the ambiguity and incompleteness issues. Based on the methods presented in this paper, we implemented the module denoted to acts clustering and inconsistency revealing.

[2]https://spacy.io

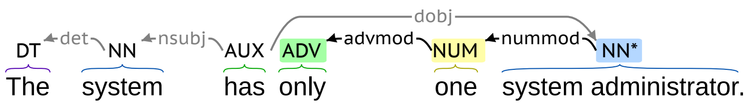

Figure 9: Consistency rule from Example 2.

6 DATA, EXPERIMENTS, AND RESULTS

To test our methods, we need the first version of textual requirements in its "raw" form. Unfortunately, there are not collections of textual requirements specifications available that would be large enough. Software companies hold them classified as confidential. So, our experimental data were of limited size. We checked the same collection of requirements that we checked in (Šenkýř and Kroha, 2019a). However, we did not find direct contradictions in these specifications. Because of that, we constructed a benchmark described below.

6.1 Data

To test our set of patterns, we needed a large set of textual requirements sentences to evaluate the precision and the recall of our method. Moreover, we needed to know the number of sentences suspicious from inconsistency before computing the numbers of true/false positive and true/false negative cases. The only way we have seen was to construct benchmark data.

We decided to focus on antonyms; negations; a combination of synonyms together with swapped roles of subject and object, passive voice, and negation; and numerically different data described in Section 3.1.1.

We used the text of the user guide describing the functionality of a known information system that is used to support stock market trading. The text volume is 1,108 pages. It consists of 14,114 sentences. We transformed the rules described in Section 3.1.1 into linguistic patterns, and we applied the patterns to find semantically overlapping sentences as candidates for inconsistency (see below).

6.1.1 The List of Antonyms

The list of antonyms used to test the inconsistency: *source—destination, first—last, all—selected, open—close, at the front—at the end, send—receive, numeric—alphabetic, high—low, valid—invalid, insert—delete, locked—unlocked, unique—duplicated, chronologically sorted—alphabetically sorted, able—unable, in only one—in one or more, private—public, at the bottom—at the top, expanded list—reduced list, upper-right corner of the window—upper-left corner of the window, undo command—redo command, enable—disable, more—less, appear—disappear, manually—automatically.*

6.1.2 The List of Negations

The list of negations used to test the inconsistency:

- simple negations of verb forms (modal verbs, e.g., *"it is the same"*, *"it is not the same"*, standard verbs, e.g., *"it exists"*, *"it does not exist"*),
- negation of the similar meaning, e.g., *"imported data can't be modified"* contra *"you can modify the imported data"*,
- complex negations of the sentence meaning, e.g., *"you can display it if you are in the Mode Checking"* contra *"you can display it at any time"*.

6.1.3 The List of Numerically Different Data

The numerical data can be given by digits, by words, or by other means, e.g., Max-Int. We test the similar sentences and generate warnings when some suspicious formulations are found. For example: *"There is an unused field that should be set to zero."* contra *"There is an unused field that should be set to Max-Int."*

Example 4: A Unique Object.

Sentence 1: The system has only one system administrator.

Sentence 2: All system administrators have the same access rights, but each of them has his/her password.

Here, *Pattern #3 (Numeric Predicate)* is matched as shown in Fig. 9. We generate a warning message, because *"only one system administrator"* in the first sentence and *"All"* and *"each of them"* (system administrators) in the second sentence contradict each other.

Pattern #4 (Determiner Predicate) is matched two times. First, it is applied in the part *"All system administrators"*. Second, it is applied in the part *"each of them"* when the co-reference to system administrators is resolved.

(End of Example)

Table 1: Clusters: Numbers, Cardinalities, and False Positives.

Cluster	**False Positives Total**		**FP -BR**		**FP -NLP**
C_1**: The same SUBJECT, VERB, and OBJECT** *#: 881 Cardinalities: 2–97*	10	=	8	+	2
C_2**: The same SUBJECT and VERB** – passive mode, without OBJECT *#: 167 Cardinalities: 2–69*	7	=	5	+	2
C_3**: The same SUBJECT** – passive mode, without OBJECT *#: 138 Cardinalities: 2–31*	2	=	2	+	0
C_4**: The same SUBJECT and OBJECT** *#: 883 Cardinalities: 2–97*	12	=	11	+	1
C_5**: The same WHAT-part and VERB** *#: 400 Cardinalities: 2–71*	0	=	0	+	0

Legend: **FP-BR** – false positives matched because of too complex sentences or sentences across different contexts, **FP-NLP** – false positives matched because of bad recognition of part-of-speech tag and/or dependency.

6.1.4 The Incomplete List of Items

We identify this source of inconsistency using the UML model that we obtained in the first phase of the processing. In the model, the attribute values are given by a list of values, but in some sentences, only a subset of them is mentioned. Often, this is a signal that a reaction to some state of the system has been forgotten. In this case, the incompleteness (Šenkýř and Kroha, 2019b) is the source of the inconsistency.

6.2 Benchmark Construction

To finish the benchmark data construction, we used the words given above in sections 6.1.2, 6.1.2, 6.1.3, and 6.1.4 to construct similar sentences (similar to the original) that should have the contradict meaning.

We interlarded the original text with our constructed sentences, and we noticed the numbers and positions of our artificially constructed sentences that are in contradiction with the original sentences to get precision and recall results when evaluating our method of inconsistency identification.

6.3 Experiments and Results

As mentioned in Section 6.1, our original text has 1,108 pages. There were 14,114 sentences found. Additionally, we constructed sentences (using words and phrase given above) that contain contradictions to some of the original sentences. In numbers, they were added:

- 21 sentences containing the antonyms as the contradiction source – *testable in clusters* C_2*,* C_3*,* C_4*, and* C_5,
- 435 sentences containing the negations as the contradiction source – *testable in all clusters*,
- 9 sentences containing the numerically different data – *testable in all clusters*,
- 32 sentences containing an incomplete list of attribute values.

Doing this, we obtained a new text having 14,611 sentences. As can be seen in the list above, some issues can be detected directly on the clusters (as proposed in Sec. 4.1), some only when compared with the internal model. After applying our method, we have got the results shown in Table 1. We recognized all issues of our added sentences, but there were, of course, some *false positives*. In the result table, we show the reasons. The first category represents false positives matched because of too complex sentences or sentences in different contexts where the current version of our *acts* and *patterns* are not enough. The second category represents false positives matched because of bad recognition of *part-of-speech tag* and/or *dependency* provided by the NLP framework.

7 CONCLUSIONS

Textual formulations of requirements specification may contain (and usually contain) some sources of contradiction that may cause inconsistency. This fact increases the software development costs because of the need for repairs during the development, tests, and maintenance.

In this paper, we described our method that supports the identification of inconsistency in textual requirements specification at the very first phase of the software development. Using linguistic modules, we find occurrences of patterns that are typical for formulations containing inconsistency. We have used synonyms, antonyms, negations, and numerically different data as the most simple linguistic sources of contradiction.

The natural language and the semantics of the described reality are too rich in possibilities how to describe it. Consequently, we speak about suspicious formulations and generate warning messages and questions on domain experts. We do not try to solve or correct the found inconsistencies automatically.

One of the limits of our method is its scalability, even though the data from our experiments (see Table 1) obtained for a text volume of 14,611 sentences are promising. Our plan is to combine our patterns concerning ambiguity, incompleteness, and inconsistency in a large industrial case study. Further, we suppose that the situation could change if we had the domain ontologies to construct consistency rules.

ACKNOWLEDGEMENTS

This research was supported by the grant of Czech Technical University in Prague No. SGS20/209/OHK3/3T/18.

REFERENCES

da Silva, A. R. and Fernandes, J. C. (2018). Variability Specification and Resolution of Textual Requirements. In *Proceedings of the 20th International Conference on Enterprise Information Systems*, pages 157–168. SciTePress – Science and Technology Publications.

de Marneffe, M.-C., Rafferty, A. N., and Manning, C. D. (2008). Finding Contradictions in Text. In Moore, J., Teufel, S., Allan, J., and Furui, S., editors, *Proceedings of ACL-08: HLT*, pages 1039–1047, Columbus, Ohio. Association for Computational Linguistics.

Fanmuy, G., Fraga, A., and Llorens, J. (2012). Requirements Verification in the Industry. In Hammami, O., Krob, D., and Voirin, J.-L., editors, *Complex Systems Design & Management*, pages 145–160, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gargiulo, F., Gigante, G., and Ficco, M. (2015). A Semantic Driven Approach for Requirements Consistency Verification. *Int. J. High Perform. Comput. Netw.*, 8(3):201–211.

Hunter, A. and Nuseibeh, B. (1998). Managing Inconsistent Specifications: Reasoning, Analysis, and Action. *ACM Trans. Softw. Eng. Methodol.*, 7(4):335–367.

Kamalrudin, M. and Sidek, S. (2015). A Review on Software Requirements Validation and Consistency Management. *International Journal of Software Engineering and Its Applications*, 9(10):39–58.

Kroha, P., Janetzko, R., and Labra, J. E. (2009). Ontologies in Checking for Inconsistency of Requirements Specification. In *2009 Third International Conference on Advances in Semantic Processing*, pages 32–37, Sliema, Malta. IEEE Computer Society Press.

Schamai, W., Helle, P., Albarello, N., Buffoni, L., and Fritzson, P. (2016). Towards the Automation of Model-Based Design Verification. *INCOSE International Symposium*, 26(1):585–599.

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. (2019a). 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.

Šenkýř, D. and Kroha, P. (2019b). Problem of Incompleteness in Textual Requirements Specification. In *Proceedings of the 14th International Conference on Software Technologies*, volume 1, pages 323–330, Porto, Portugal. INSTICC, SCITEPRESS – Science and Technology Publications.

Šenkýř, D. and Kroha, P. (2021). Problem of Inconsistency and Default Consistency Rules. Prepared to be submitted.

Šimko, V., Kroha, P., and Hnětynka, P. (Prague, 2013). Implemented Domain Model Generation. Technical Report No. D3S-TR-2013-03, Department of Distributed and Dependable Systems, Faculty of Mathematics and Physics, Charles University.

Spanoudakis, G. and Finkelstein, A. (1998). A Semi-automatic Process of Identifying Overlaps and Inconsistencies between Requirements Specifications. In Rolland, C. and Grosz, G., editors, *OOIS'98*, pages 405–424, London. Springer London.

Spanoudakis, G. and Zisman, A. (2001). Inconsistency Management in Software Engineering: Survey and Open Research Issues. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, pages 329–380. World Scientific.

Torre, D., Labiche, Y., Genero, M., and Elaasar, M. (2018). A Systematic Identification of Consistency Rules for UML Diagrams. *Journal of Systems and Software*, 144:121–142.