

# Automated Requirement Formalization Using Product Design Specifications

Robin Gröpler<sup>1</sup>, Libin Kutty<sup>1</sup>, Viju Sudhi<sup>1</sup> and Daran Smalley<sup>2</sup>

<sup>1</sup>*ifak - Institut für Automation und Kommunikation e.V., Werner-Heisenberg-Str. 1, 39106 Magdeburg, Germany*

<sup>2</sup>*ALSTOM, Östra Ringvägen 2, 721 73 Västerås, Sweden*

## Abstract

Assuring the quality of complex and highly configurable software systems is a demanding and time-consuming process. Especially for safety-critical systems, extensive testing based on requirements is necessary. Methods for model-based test automation in agile software development offer the possibility to overcome these difficulties. However, it is still a major effort to create formal models from functional requirements in natural language on a large scale. In this paper, we present and evaluate automated support for the requirements formalization process to reduce cost and effort. We present a new approach based on Natural Language Processing (NLP) and textual similarity using requirements and product design specifications to generate human- and machine-readable models. The method is evaluated on an industrial use case from the railway domain. The recommended requirement models for the considered propulsion system show an average accuracy of more than 90% and an exact match of the entire models of about 55%. These results show that our approach can support the requirements formalization process, which can be further used for test case generation and execution, as well as for requirements and design verification.

## Keywords

Requirements engineering, requirements modeling, natural language processing, textual similarity

## 1. Introduction

Increasingly complex and highly configurable software systems also increase the effort required for their quality assurance. The rapid and simultaneously high-quality development of industrial software products demands an increasingly effective test process. Especially for safety-critical systems, such as in the automotive and railway domains, extensive testing based on requirements is necessary. However, any manual processing, such as requirements verification and test generation, from textual requirements is time-consuming and error-prone, and also requires a lot of expert knowledge. Methods for model-based test automation in agile software development pursue the goal to overcome these difficulties [1]. Formal models serve as the basis for automating a large number of further process steps.


---

In: J. Fischbach, N. Condori-Fernández, J. Doerr, M. Ruiz, J.-P. Steghöfer, L. Pasquale, A. Zisman, R. Guizzardi, J. Horkoff, A. Perini, A. Susi, M. Daneva, A. Herrmann, K. Schneider, P. Mennig, F. Dalpiaz, D. Dell'Anna, S. Kopczyńska, L. Montgomery, A.G. Darby, P. Sawyer (eds.): *REFSQ 2022 Joint Proceedings of the Co-Located Events, NLP4RE, Aston, Birmingham, UK, 21-03-2022*

✉ robin.groeppler@ifak.eu (R. Gröpler); libinjohn26@gmail.com (L. Kutty); vjusudhi@gmail.com (V. Sudhi); daran.smalley@alstomgroup.com (D. Smalley)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

In previous works, a model-based testing tool chain has been developed to enable efficient test processes [2, 3]. Methods for model synthesis, model-based test generation and prioritization are used to systematically and efficiently create a test suite that contains suitable test cases. This approach is based on behavioral requirements that serve as input for further processing. The only time-consuming manual step is the creation of requirement models from functional requirements in natural language.

Recent advances in natural language processing (NLP) show promising results for supporting a wide range of requirements analysis tasks [4]. Therefore, NLP techniques show a growing interest in automating various software testing activities such as model and test case generation. A considerable amount of NLP approaches and tools have been investigated in recent years aiming to generate models or test cases from underlying requirements documents [5, 6].

However, due to the shortcomings of natural language, most of the existing NL-based approaches generate abstract models and test cases, but lack information about real entities, their relations and design [6]. The links to real entities such as components, signals and parameters are usually added only during test execution, which in turn is an error-prone and time-consuming manual process. At this late stage of the testing process, issues have to be resolved, e.g., that parts of the requirements are not verifiable. This makes the model-based approach somewhat inefficient, since the requirements and the design specification then have to be revised again. Besides mapping abstract to real entities, there are usually many specific details in the system design from which the requirement formalization process would benefit. The system architecture not only describes the structure, but also includes architectural design decisions and is therefore closely related to the requirements [7].

In this work, we want to take into consideration product design specifications for a much more precise requirements formalization process. Our new approach utilizes NLP techniques to automatically generate requirement models from natural language requirements and design specifications. We perform textual similarity and contradiction analysis between the requirements and entity descriptions using classical to modern NLP algorithms. The generated models are represented in a simple, self-created, machine- and human-readable language. The *main contributions* of this work are i) a new approach for requirement formalization using product design specifications and ii) the evaluation of various algorithms on an industrial use case. The integration of information from the design specifications at this early stage of the testing process shall provide much faster feedback to the requirements or test engineer whether the requirements are verifiable and correctly designed.

## 2. Related work

There is a wide range of sophisticated methods and tools for *requirement formalization*, see e.g. the surveys of Zhao et al. [4], Buzhinsky [5] or Brunello et al. [8]. Recently, Giannakopoulou et al. [9] proposed a structured natural language called FRETISH using semantic templates, which has been evaluated in several NASA projects. These methods focus purely on the requirements texts without considering further information about the system architecture.

There are only a few works that take *architecture and design specifications* into account during the formalization of requirements. Bernaerts et al. [10] addressed the early integration of the

design process, but focus more on temporal logic while assuming a manual interpretation of natural language texts. Stachtari et al. [11] addressed the early assurance of consistency between the requirements and design correctness using a pattern-based approach. Note that in a previous study, we pointed out that a predefined list of semantic entities is helpful to generate more accurate models from requirements [2]. Wang et al. [12] developed a pattern-based approach called PASER for checking the consistency between the generated models and the implementation, i.e., considering the design in the stage of the process model. During test execution, many authors use a mapping table or test case specifications [6]. However, all of these approaches require large manual effort and expert knowledge of system architecture and design to create such documents.

The use of *architecture and design specifications* has been investigated for various other requirements analysis tasks. Leitão and Medeiros [13] developed an NLP-based method that can extract and associate components from product design specifications and system requirements. Sharma et al. [14] proposed a recommender system for selecting a suitable architectural pattern for a given set of software requirements using textual entailment. Niklas et al. [15] developed an approach for checking the consistency of design specifications against natural language requirements based on noun extraction and graph-based modeling. Yet, none of these approaches can be used for the formalization of requirements.

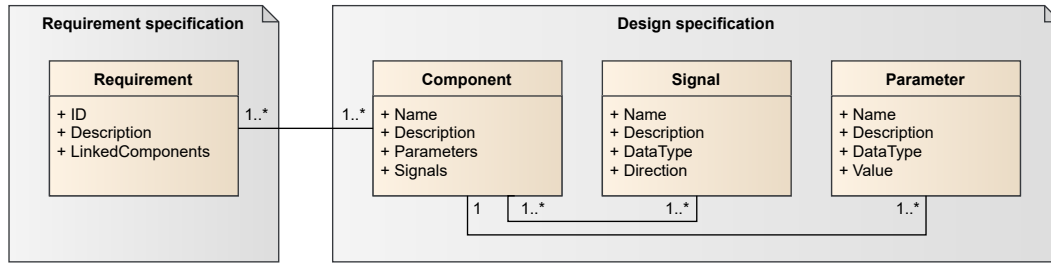
The application of *similarity approaches* is widely used in requirements analysis. For example, several techniques for identifying requirement duplicates and interdependencies have been studied [16, 17]. Furthermore, it was shown that measuring the similarity between new and old requirements of software projects helps to identify reusable software components such as design, coding and test cases [18]. Recently, Abbas et al. [19] investigated the semantic similarity of customer requirements to generate reuse recommendations for software product line assets.

However, most previous work has focused on the level of the entire requirement, which has the advantage that it can be processed at once and most NLP techniques are directly applicable. In contrast, our approach goes into much more detail on the individual entities and their dependencies between the requirements and design descriptions. In addition, not only similarity but also contradiction is studied to identify specific properties of the requirement model. To the best of our knowledge, the application of NLP-based methods for automated requirements formalization using requirements and design specifications has not yet been considered in the literature.

### 3. Use case

In order to conduct experiments and evaluate our approach, we consider a use case from the rail industry. We use the data of the Propulsion Control (PPC) system in Bombardier Transportation, an Alstom Group Company. The PPC is part of a large, complex, safety-critical system. It handles the control of the entire propulsion system, including both control software as well as the electrical functions.

The requirements are written in textual format in the requirements management tool IBM DOORS. They are written in English and may contain several sentences. They do not follow a prescribed format in order not to focus on syntax when writing them. To meet the standards



**Figure 1:** Class diagram of the input data provided in the documents.

of Safety Integrity Level 2 (SIL2), a design document is written in textual format and is also handled in DOORS. The requirements and the design document are created by hand and can be manually linked. The system architecture and software modules are modeled as blocks in Matlab Simulink using a model-based design approach [20]. The two underlying documents for our study are provided as exports from DOORS and Simulink.

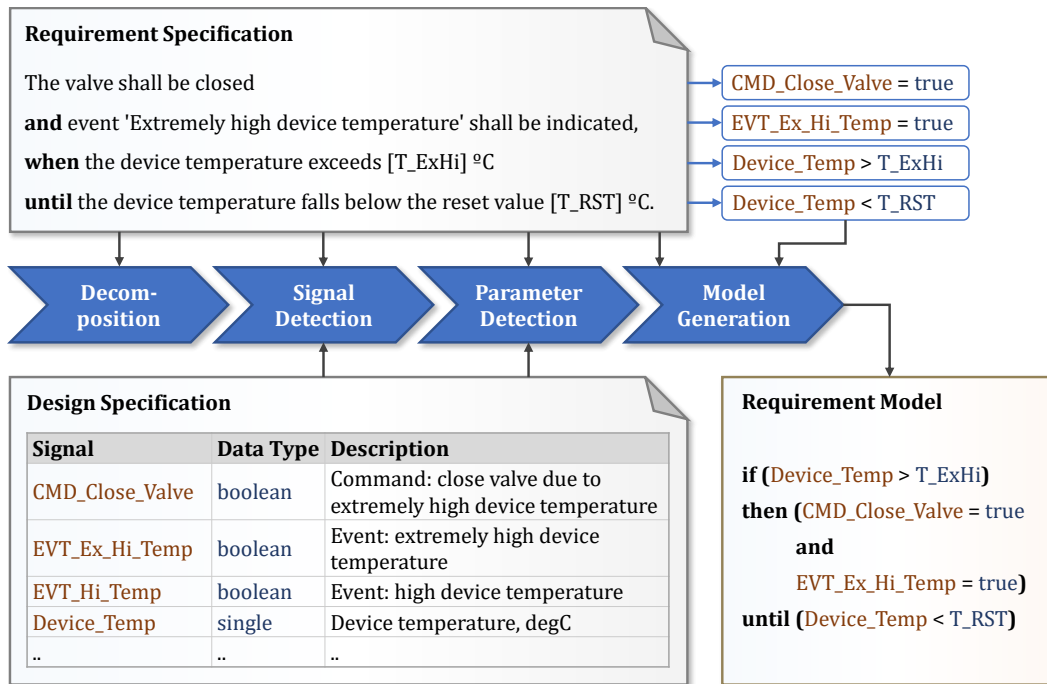
We assume the following tool and use case specific characteristics of the requirements and design specifications, cf. Figure 1: *i*) components are not mentioned in the textual requirement descriptions but are defined in the design specification and linked to the requirements, *ii*) all signals and parameters are defined with textual descriptions in the design specification and linked to the corresponding components, and *iii*) requirement descriptions have a clear and consistent structure. In particular, we assume that a short description in natural language is available for each entity. This assumption should hold true in most industrial product developments since requirements engineers also need to identify entities by some textual description. This information can be written inside the implemented code, in the company's internal standardization documentation, or more structured as a table in a design specification document.

## 4. Methodology

In the following, we investigate suitable NLP methods to automate the process of creating requirement models from the requirements and design specification. Figure 2 demonstrates the pipeline of our approach using an exemplary requirement with related signal descriptions. Note that the requirement is shown somewhat generalized, as the data we use is confidential.

The process can be divided into the following steps. The requirements in natural language and the design specification serve as inputs to the pipeline. The requirement is initially decomposed into sub-requirement clauses (Section 4.1). Each clause is compared to the signal descriptions in the design specification to identify the signals (Section 4.2) and then the corresponding parameters are identified (Section 4.3). A relation is formed for each clause, which is then translated into a logical structure and finally into a requirement model (Section 4.4).

The ground truth for the individual steps was created manually and reviewed by a requirements engineer with expert knowledge. While presenting the methodology, we show experimental results for the individual steps and discuss which methods are most suitable.



**Figure 2:** Exemplary requirement with an overview of the pipeline.

#### 4.1. Decomposition

It is imperative to initially decompose the long, complex industrial requirements into shorter requirement clauses that describe elementary instructions individually. This process requires understanding both the structure (syntax) and meaning (semantics) of the requirement text. In our first trials to decompose the requirements, we used a naive keyword-based approach. The conjunctions for introducing conditional clauses like *if*, *when*, *while* and *until* and for connecting clauses like *and* and *or* were identified in the span of the requirement text and were considered the boundary of clauses. In the considered use case with well structured requirements, there were only a few cases where this approach did not work. For example, the phrases "*between sensor 1 and 2*" or "*less than or equal*" should not be considered a boundary despite the presence of a conjunction. To decompose such requirements correctly, we considered the syntactic dependencies<sup>1</sup> of the requirement text (e.g., using the *ancestor* tokens of the conjunctions) in addition to the keywords. Fig. 2 illustrates the decomposition of a requirement (by line breaks) from our use case. Our use case demands only a simple decomposition algorithm, as proposed here, to work correctly. However, this could be extended to more linguistic patterns as proposed in [21] or to more sophisticated algorithms such as proposed in [22].

For the further steps in our pipeline, decomposition into requirement clauses is best suited. A shorter chunk of text (with just the noun phrases) leads to a loss of information and a longer clause (a whole sentence) leads to erroneous detection of signals and parameters.

<sup>1</sup>using spaCy, <https://spacy.io/>

**Table 1**  
Evaluation results for signal detection

Class	Method	Accuracy
Classical methods	<b>Term Frequency (TF)</b>	78.5%
	<b>Term Frequency with stemming (TF-stem)</b>	86.0%
String matching	<b>FuzzyWuzzy</b> Partial Ratio	67.3%
	<b>FuzzyWuzzy</b> Token Set Ratio (Fuzz-TSR)	85.0%
Static embeddings	<b>GloVe</b>	74.0%
	<b>fastText</b>	57.0%
Contextual embeddings	<b>Sentence-BERT (SBERT)</b>	90.7%
Ensembles	<b>TF-stem + Fuzz-TSR</b>	90.7%
	<b>SBERT + Fuzz-TSR</b>	<b>91.6%</b>

## 4.2. Signal detection

Once the requirement is decomposed, we compute the cosine similarity between the requirement clause and all the signal descriptions in the design specification. The signal with the highest similarity score is then retrieved as the most relevant. To obtain a vector representation for each requirement clause and signal description, we apply classical techniques including Term Frequency (TF), static word embeddings including GloVe<sup>2</sup> and fastText<sup>3</sup> and contextual embeddings from Sentence-BERT<sup>4</sup> (SBERT). We also use FuzzyWuzzy<sup>5</sup>, which looks for partial (or inexact) matches between sentence pairs. We also consider ensembles of these methods by taking the average of the individual similarity scores.

The evaluation results for signal detection using a total of 107 clauses and 207 signals from the use case are shown in Table 1. We use TF instead of TF-IDF as we observed that the IDF weights do not change the result significantly (probably due to the shortness of texts). In agreement with this argument, we found that removing stop words deteriorated the similarity scores. However, stemming<sup>6</sup> the text before vectorizing with TF gives better results. Pre-trained static embedding models performed hardly well for detecting signals. We observed that these models have difficulty distinguishing between, e.g., "*extremely high temperature*" and "*too high temperature*" within the signal description. Though the SBERT model performs very well, there are also some cases where the model could not distinguish between "*device on fault*" and "*device off fault*". This could be attributed to the modality of their training, which aims to bring embeddings of similar words in their representation space as close as possible. As a result, the model gives very high cosine similarity scores for such pairs and occasionally detects an incorrect signal. Though FuzzyWuzzy individually did not perform as well as the other methods, when combined with the relatively better performing TF-stem and SBERT models, the best results were obtained.

<sup>2</sup>using *glove.42B.300d* from <https://nlp.stanford.edu/projects/glove/>

<sup>3</sup>using *wiki-news-300d-1M.vec* from <https://fasttext.cc/docs/en/english-vectors.html>

<sup>4</sup>using *paraphrase-distilroberta-base-v1* from <https://github.com/UKPLab/sentence-transformers>

<sup>5</sup><https://github.com/seatgeek/fuzzywuzzy>

<sup>6</sup>using Porter stemmer from NLTK, [https://www.nltk.org/\\_modules/nltk/stem/porter.html](https://www.nltk.org/_modules/nltk/stem/porter.html)



**Table 2**

Evaluation results for parameter detection with boolean data type

Class	Method	TPR	TNR	Balanced Accuracy
Antonyms	<b>WordNet</b>	50.7%	33.3%	42.1%
	<b>MoE-ASD</b> trained on all POS tags	55.2%	61.1%	58.2%
Paraphrase	<b>Term-Frequency</b> with stemming	50.7%	55.6%	53.2%
	<b>Fuzzy Wuzzy</b> Token Set Ratio	71.6%	22.2%	46.9%
	<b>Sentence-BERT</b> (SBERT)	73.1%	66.7%	69.9%
Inference	<b>Sentiment Analysis</b>	79.1%	50.0%	64.6%
	<b>Textual Entailment</b> (TE)	88.1%	61.1%	74.6%
	<b>Textual Entailment</b> with negation rule (TE-neg)	<b>95.5%</b>	<b>72.2%</b>	<b>83.9%</b>

### 4.3. Parameter detection

After detecting the signals, we need to determine the parameters that are assigned or compared to the signal values. When parameter names are used within the requirement (e.g.,  $T_{ExHi}$ ), they can be easily identified by simple pattern matching in the design specification. In many cases, however, the parameters are of boolean data type and can only be identified by a semantic comparison between the two textual statements. In these cases, we need to determine whether the requirement clause agrees or contradicts with the corresponding signal description to obtain the parameter. For example, the requirement clause "*The valve shall be opened*" and the signal description "*close valve ..*" contradict each other and the parameter is set to *false*. Conversely, if the signal description agrees with the requirement clause, the parameter should be detected as *true*. Such NLP tasks for detecting contradictions are known to be difficult [23].

Our first trial was antonym synonym detection, where we checked whether the verb in the requirement clause falls into the *antonym sets* in WordNet<sup>7</sup> of the verb in the signal description. Similar to this approach, we used the model of Mixture-of-Experts for Antonym-Synonym Discrimination (MoE-ASD) [24]. While these methods handle pairs like "*activated*" and "*deactivated*" considerably well, they fail for phrasal negations like "*not activated*". To overcome this issue and avoid the ambiguity of model inferences that depend only on the verb, we trained MoE-ASD with all the POS tags and tried to infer with the sentence representation as input.

We also utilize the similarity scores from the signal detection step and identify the parameter as *true* if the scores are above a threshold (empirically set to 0.6) and *false* otherwise. This is motivated by the idea that when paraphrasing a sentence into another, one could identify whether it agrees or contradicts with the other sentence.

Since textual inference methods can reliably classify a pair of sentences as either agreement or contradiction, we also resort to sentiment analysis<sup>8</sup> to assess the sentiment of the pair and determine the parameter from it. We also tried to infer the entailment relationship using a pre-trained model<sup>9</sup> trained with 3 classification labels. We considered the softmax probabilities for

<sup>7</sup><https://wordnet.princeton.edu/>

<sup>8</sup>using GLoVe-LSTM from <https://demo.allennlp.org/sentiment-analysis/glove-sentiment-analysis>

<sup>9</sup>using ELMo-based Decomposable Attention from <https://demo.allennlp.org/textual-entailment/elmo-snli>

the labels *agreement* and *contradiction* and predicted the parameter accordingly. In neutral cases, we employed a negation rule that checks for the presence of words that identify a negation.

Table 2 shows our evaluation results for parameter detection using 85 signals with boolean parameters, of which 67 are actually *true* and 18 are *false*. It is evident from the table that textual entailment combined with the negation rule gives the best result.

#### 4.4. Model generation

Once the signals and parameters are identified for each requirement clause, we formulate a relation (of the form *Signal-Operator-Parameter*) for each of these clauses. We rely on a dictionary-based approach<sup>10</sup> to identify comparison operators (if any) in the clauses. In Fig. 2 the relations are illustrated for the exemplary requirement. To generate requirement models from these relations, we introduce a domain-specific language (DSL) with abstract logical blocks. It maps the relations coming from a conditional clause (*if/when/while*) to an *if-block*, those coming from main clauses to a *then-block* and those coming from an *until*-clause to an *until-block*. Conjunctions (*and/or*) identified between relations are also accommodated in these DSL blocks. Though this mapping appears rather trivial, our aim is to make this translation simple and flexible, so that ways are open for integration with other sophisticated languages. The resulting models can also be further transformed into Matlab Simulink models or UML sequence diagrams, depending on what the end user desires.

### 5. Evaluation

To evaluate the entire pipeline against the 31 requirements from our use case, we combine all the individual steps from above. We take the requirement texts as input and generate requirement models with the help of the design specification.

The evaluation results are shown in Table 3. The average accuracies are calculated as follows. For each requirement, we calculate the percentage of correctly identified relations. Similarly, we calculate the accuracy of the logic by counting the correctly connected relations with logical conjunctions (*and/or*) and the correctly assigned relations to the logical blocks (*if/then/until*). For the accuracy of the model, we count all correctly identified signals, parameters, relations, and logic. Then, the macro-average over all requirements is calculated and shown in Table 3. The exact match counts the percentage of completely correct models. While this is a tough metric, it shows the extent to which the requirements formalization process can be fully automated.

We observe the best results with the two best ensembles for signal detection together with the best method (TE-neg) for parameter detection. This combination yields an average accuracy of 90.7% in evaluating the model across all requirements. The rather low values for exact match of 54.8% reflect the toughness of the metric, which invalidates a requirement model even if a single constituent (mostly one parameter in our case) is misidentified, although the remaining constituents are correctly identified. Some generalized resources of this paper can be found on GitHub<sup>11</sup>.

---

<sup>10</sup>using Roget's Thesaurus, <http://www.roget.org/scripts/hier.php?class=I&division=0&section=III>.

<sup>11</sup>[https://github.com/ifak-prototypes/nlp\\_reform](https://github.com/ifak-prototypes/nlp_reform)



**Table 3**

Evaluation results for the whole pipeline on the propulsion system

Methods		Average Accuracy			Exact Match
Signal detection	Param. detection	Relations	Logic	Model	Model
<b>TF-stem + Fuzz-TSR</b>	<b>SBERT</b>	69.6%	69.9%	86.4%	25.8%
<b>SBERT + Fuzz-TSR</b>	<b>SBERT</b>	70.4%	69.9%	86.4%	25.8%
<b>TF-stem + Fuzz-TSR</b>	<b>TE-neg</b>	84.4%	82.8%	<b>90.7%</b>	<b>54.8%</b>
<b>SBERT + Fuzz-TSR</b>	<b>TE-neg</b>	84.4%	82.8%	<b>90.7%</b>	<b>54.8%</b>

Although our approach shows promising results, it has several limitations. One of the main limitations of our pipeline could be the simple decomposition algorithm that works for our specific requirements. Therefore we referred to more elaborated algorithms when needed. The model language is also very limited and contains only the basic behavioral elements. Finally, we assume that links are provided between artifacts, especially to the components.

## 6. Conclusion and outlook

In this work, we presented an NLP-based approach for automated requirements formalization using natural language requirements and design specifications. We investigated various NLP methods for the individual steps of our pipeline and evaluated them on an industrial use case from the railway domain. We have shown that for signal detection we obtain the best results using Sentence-BERT combined with FuzzyWuzzy-TSR, and for (boolean) parameter detection using Textual Entailment supported by a negation rule. When evaluating the entire pipeline, we found that the requirement models generated using the combination of the two aforementioned methods yield the highest average accuracy of more than 90% and an exact match of about 55%. These results show that our approach can highly automate the process of requirements formalization, which can support the requirements engineer in e.g. requirements verification and test case generation.

In the future, we plan to integrate this approach into our model-based testing pipeline. Furthermore, there are several features that could improve our methodology, such as handling non-functional properties (like durations), prioritizing methods with lower execution time, domain-specific pre-training and fine-tuning of available models (like SBERT), or learning from corrected predictions of a domain expert (online learning). Another interesting enhancement would be to support the requirements design process and the manual creation of the design specification by providing recommendations for related components and signals in advance.

## Acknowledgments

This research was funded by the ITEA3 project XIVT. The German partner was funded by the BMBF (grant no. 01IS18059E). We thank Bombardier Transportation, an Alstom Group Company, for providing an industrial use case for the evaluation of the presented method.

## References

- [1] M. Utting, B. Legeard, *Practical Model-Based Testing: A Tools Approach*, Elsevier, 2010.
- [2] R. Gröpler, V. Sudhi, E. J. Calleja García, A. Bergmann, NLP-based Requirements Formalization for Automatic Test Case Generation, in: *CS&P'21*, 2021, pp. 18–30.
- [3] M. Reider, S. Magnus, J. Krause, Feature-based testing by using model synthesis, test generation and parameterizable test prioritization, in: *ICSTW*, 2018, pp. 130–137.
- [4] L. Zhao, W. Alhoshan, A. Ferrari, K. Letsholo, M. Ajagbe, E.-V. Chioasca, R. T. Batista-Navarro, Natural Language Processing (NLP) for Requirements Engineering (RE): A Systematic Mapping Study, *ACM Computing Surveys* (2020).
- [5] I. Buzhinsky, Formalization of natural language requirements into temporal logics: a survey, in: *INDIN*, 2019, pp. 400–406.
- [6] C. Wang, F. Pastore, A. Goknil, L. Briand, Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach, *TSE* (2020) 1–38.
- [7] R. C. De Boer, H. Van Vliet, On the similarity between requirements and architecture, *Journal of Systems and Software* 82 (2009) 544–550.
- [8] A. Brunello, A. Montanari, M. Reynolds, Synthesis of LTL formulas from natural language texts: State of the art and research directions, in: *TIME*, 2019, pp. 17:1–17:19.
- [9] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Schumann, Automated formalization of structured natural language requirements, *Information and Software Technology* 137 (2021) 106590.
- [10] M. Bernaerts, B. Oakes, K. Vanherpen, B. Aelvoet, H. Vangheluwe, J. Denil, Validating industrial requirements with a contract-based approach, in: *MODELS-C*, 2019, pp. 18–27.
- [11] E. Stachtari, A. Mavridou, P. Katsaros, S. Bliudze, J. Sifakis, Early validation of system requirements and design through correctness-by-construction, *J. Syst. Softw.* 145 (2018) 52–78.
- [12] Y. Wang, T. Wang, J. Sun, PASER: a pattern-based approach to service requirements analysis, *Int. J. Softw. Eng. Knowl. Eng.* 29 (2019) 547–576.
- [13] V. Leitão, I. Medeiros, SRXCRM: Discovering Association Rules Between System Requirements and Product Specifications, in: *NLP4RE'21*, 2021, pp. 1–9.
- [14] S. Sharma, B. Sodhi, APR: architectural pattern recommender, in: *SAC'17*, 2017, pp. 1225–1230.
- [15] K. Niklas, S. Gärtner, K. Schneider, Consistency checks of design specifications against requirements using graph-based linguistic analysis, in: *SAC'16*, 2016, pp. 1546–1549.
- [16] J. Natt, B. Regnell, P. Carlshamre, M. Andersson, J. Karlsson, Evaluating automated support for requirements similarity analysis in market-driven development, in: *REFSQ'01*, 2001, pp. 190–201.
- [17] D. Falessi, G. Cantone, G. Canfora, Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques, *TSE* 39 (2011) 18–44.
- [18] M. Ilyas, J. Küng, A comparative analysis of similarity measurement techniques through SimReq framework, in: *FIT'09*, 2009, pp. 1–6.
- [19] M. Abbas, M. Saadatmand, E. Enoiu, D. Sundamark, C. Lindskog, Automated reuse recommendation of product line assets based on natural language requirements, in: *ICSR*, 2020, pp. 173–189.
- [20] E. Simonson, A Model-Based Design Adoption Story from Bombardier Transportation, in: *MATLAB EXPO 2018 Sweden*, 2018, pp. 1–16.
- [21] M. Liu, X. Peng, A. Marcus, C. Treude, X. Bai, G. Lyu, J. Xie, X. Zhang, Learning-based extraction of first-order logic representations of API directives, in: *ESEC/FSE*, 2021, pp. 491–502.
- [22] J. Malin, C. Millward, F. Gomez, D. Throop, Semantic annotation of aerospace problem reports to support text mining, *IEEE Intelligent Systems* 25 (2010) 20–26.
- [23] R. Rivera, P. Martínez, The Impact of Pretrained Language Models on Negation and Speculation Detection in Cross-Lingual Medical Text: Comparative Study, *JMIR Medical Informatics* 8 (2020).
- [24] Z. Xie, N. Zeng, A Mixture-of-Experts Model for Antonym-Synonym Discrimination, in: *Proceedings of ACL-IJCNLP*, 2021, pp. 558–564.