TAPHSIR: Towards AnaPHoric Ambiguity Detection and ReSolution In Requirements

Saad Ezzini
University of Luxembourg
Luxembourg
saad.ezzini@uni.lu

Sallam Abualhaija
University of Luxembourg
Luxembourg
sallam.abualhaija@uni.lu

Chetan Arora
Deakin University
Australia
chetan.arora@deakin.edu.au

Mehrdad Sabetzadeh
University of Ottawa
Canada
m.sabetzadeh@uottawa.ca

ABSTRACT

We introduce TAPHSIR – a tool for anaphoric ambiguity detection and anaphora resolution in requirements. TAPHSIR facilities reviewing the use of pronouns in a requirements specification and revising those pronouns that can lead to misunderstandings during the development process. To this end, TAPHSIR detects the requirements which have potential anaphoric ambiguity and further attempts interpreting anaphora occurrences automatically. TAPHSIR employs a hybrid solution composed of an ambiguity detection solution based on machine learning and an anaphora resolution solution based on a variant of the BERT language model. Given a requirements specification, TAPHSIR decides for each pronoun occurrence in the specification whether the pronoun is ambiguous or unambiguous, and further provides an automatic interpretation for the pronoun. The output generated by TAPHSIR can be easily reviewed and validated by requirements engineers. TAPHSIR is publicly available on Zenodo (https://doi.org/10.5281/zenodo.5902117).

CCS CONCEPTS

• **Software and its engineering** → **Requirements analysis**; • **Computing methodologies** → **Machine learning**; **Natural language processing**.

KEYWORDS

Requirements Engineering, Natural-language Requirements, Ambiguity, Natural Language Processing, Machine Learning, BERT

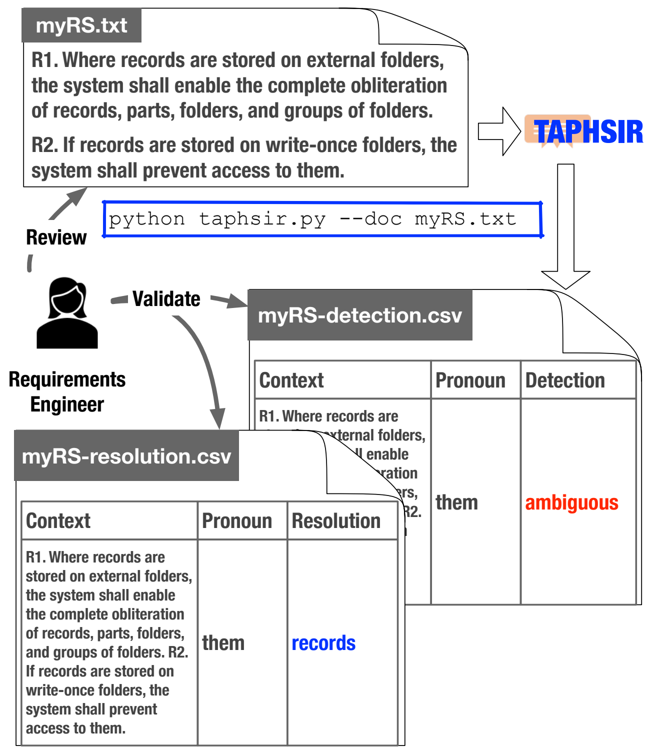
ACM Reference Format:
Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. TAPHSIR: **T**owards **A**na**PH**oric Ambiguity Detection and Re**S**olution **I**n **R**equirements. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3540250.3558928

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3558928

1 INTRODUCTION

The overall success of a project depends to a large extent on the quality of requirements [4, 5, 10]. In particular, ensuring the precision and consistency of requirements is paramount for avoiding major development risks such as time and budget overruns, failure to meet customers' needs, and systems that are not trustworthy. The requirements quality challenge is exacerbated by the fact that requirements are often written in natural language [17]. Although natural language facilitates communication among different stakeholders, textual requirements are highly prone to ambiguity. At an early stage of software development, requirements engineers spend considerable time and effort inspecting requirements specifications (RSs) to identify various quality issues such as incompleteness, inconsistency and ambiguity. Doing such inspections entirely manually is not only time-consuming but can also be error-prone, since engineers may overlook *unacknowledged ambiguity*. Ambiguity is unacknowledged when different individuals have diverging interpretations for the same requirement, and yet, each individual is confident about their own interpretation. In such cases, the requirement from the perspective of each individual is regarded as unambiguous and thus not flagged for further discussion. Compared to acknowledged ambiguity that is often discussed and resolved during inspection sessions, unacknowledged ambiguity may propagate to later stages of development and lead to serious problems due to unconscious misunderstandings.

In this paper, we propose the tool *TAPHSIR*, standing for **T**owards **A**na**ph**oric Ambiguity Detection and Re**s**olution **i**n **R**equirements. In Arabic, TAPHSIR means "interpretation". TAPHSIR focuses on pronominal anaphoric ambiguity, an ambiguity type that has been explored only to a limited extent in requirements engineering (RE) [6, 22]. There are no existing tools in RE to handle anaphoric ambiguity, although this type of ambiguity is prevalent in NL requirements: it is estimated that up to 20% of industrial requirements may suffer from anaphoric ambiguity [6, 19]. TAPHSIR implements the best solution emerging from our multi-solution study of anaphoric ambiguity in natural-language requirements, published in a technical paper [3] at the 44th International Conference on Software Engineering (ICSE 2022). This best solution is a hybrid one, where feature-based machine learning (ML) is used for detecting anaphoric ambiguity and a large-scale language model (LM) from the BERT family is used for anaphora resolution.



R1. Where records are stored on external folders, the system shall enable the complete obliteration of records, parts, folders, and groups of folders.

R2. If records are stored on write-once folders, the system shall prevent access to them.

```
python taphsir.py --doc myRS.txt
```

Context	Pronoun	Detection
R1. Where records are ...xternal folders, ...ll enable ...ration ...rs, ...R2.	them	ambiguous

Context	Pronoun	Resolution
R1. Where records are stored on external folders, the system shall enable the complete obliteration of records, parts, folders, and groups of folders. R2. If records are stored on write-once folders, the system shall prevent access to them.	them	records

Figure 1: Application Example of TAPHSIR.

Compared to our earlier technical paper [3], we present in this current paper an in-depth, practitioner-oriented description of TAPHSIR, elaborating the tool's architecture and its engineering as well as how end-users can use the tool. We further discuss the accuracy of TAPHSIR in detecting anaphoric ambiguity (including unacknowledged cases) and resolving anaphora.

TAPHSIR aims to reduce the time and effort that requirements engineers spend inspecting the use of pronouns in an RS. To illustrate, consider the example in Figure 1. The figure shows a requirements engineer reviewing the requirements in the file "myRS.txt" and using TAPHSIR for automated analysis of pronominal anaphora in that RS. The pronoun "them" in R2 contains anaphoric ambiguity since it is not clear whether the pronoun refers to the *write-once folders* (in R2), *records* only (in R1), or *records, parts, folders and groups of folders* altogether (in R1). Deciding about the exact interpretation has an impact on properly implementing the requirement. TAPHSIR defines a *context* for each pronoun occurrence. This context is composed of the requirement in which the pronoun occurs and the preceding requirement. Within this context, the tool identifies all noun phrases (NPs) preceding the pronoun as candidate antecedents [14]. In our example, TAPHSIR will consider, in addition to those mentioned above, the following candidate antecedents: *access, obliteration, system.* TAPHSIR then goes through different steps as we explain in the next section, and produces an output file ("myRS.csv" in Figure 1). This output lists all pronoun occurrences in the input RS, and provides both the detection decision as well as the resolution result. We note that TAPHSIR can recommend a resolution also for those pronouns that are marked as ambiguous, since it applies two separate solutions for detection and resolution. Running TAPHSIR in this example requires ≈22.5 seconds to produce the results [3].

In the remainder of this tool demonstration paper, we outline TAPHSIR's main components. We further discuss through the lens of unacknowledged ambiguity the evaluation of TAPHSIR on a manually curated dataset (*DAMIR* [3]).

2 TOOL ARCHITECTURE

TAPHSIR is a usable prototype tool for anaphoric ambiguity handling. The tool realizes a technical solution that resulted from an empirical examination of several alternative solutions [3]. Figure 2 shows an overview of TAPHSIR architecture. The tool is implemented in Python 3.8 [20]. Below, we discuss an end-to-end application of the tool going through steps 1 – 7 of Figure 2.

2.1 Preparation

Prior to using the tool, the user needs to perform some preliminary setup. To do so, one can type in the following on the command line:

```
python pip install -r libraries.txt
python -m spacy download en_core_web_sm
```

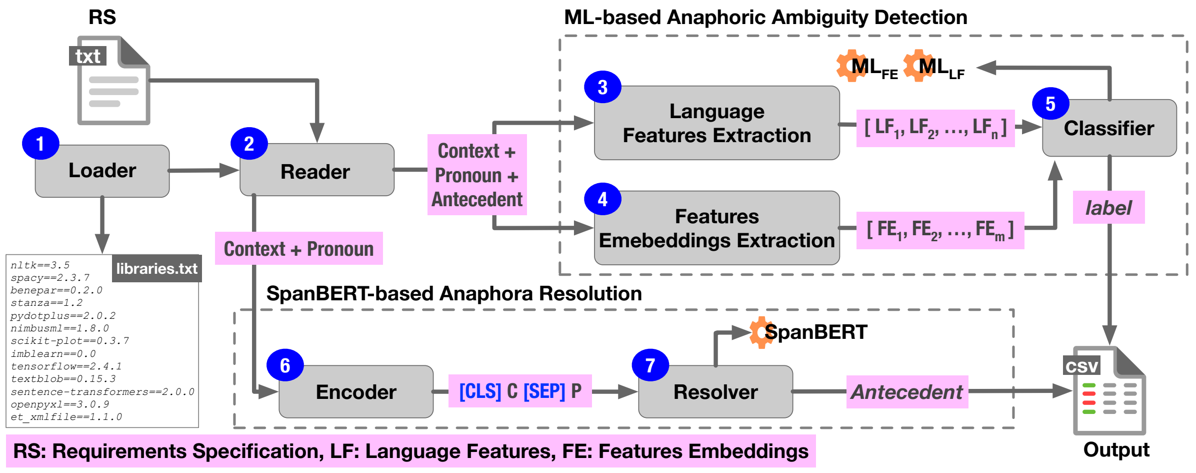
The first command installs the required libraries, and the second one downloads *en_core_web_sm* which is needed for operationalizing the natural language processing pipeline in SpaCy. To be able to apply the tool, the user further needs to ensure that the input file is in the right format. TAPHSIR expects as input a text file (with the extension **.txt*) containing a set of requirements (or sentences).

2.2 Reader

This step parses the text of the input requirements specification, preprocesses it using an NLP pipeline, and identifies the requirements that should be subject to anaphoric ambiguity analysis. The NLP pipeline consists of the following seven modules illustrated in Figure 2: (i) tokenizer splits the input text into tokens, (ii) sentence splitter demarcates the sentences in the text, (iii) part-of-speech tagger (POS) assigns a POS tag for each token, (iv) lemmatizer identifies the canonical form of a token, (v) constituency parser identifies the structural units of sentences, (vi) dependency parser defines the grammatical dependencies between the tokens in sentences, and (vii) semantic parser extracts information about words' meanings.

The output of this step is a set of *triples,* each of which includes a (i) a *pronoun* occurrence, (ii) *context* defined as the requirement in which the pronoun occurs and a preceding requirement (recall from Section 1, and (iii) a likely *antecedent* to that pronoun occurrence. The number of triples depends on the number of likely antecedents. In Figure 1, there are three possible antecedents as discussed in Section 1, namely "records, parts, folders and groups of folders", "records", and "write-once folders". Following this, this steps generates three triples associated with the pronoun occurrence "them", where each triple includes one possible antecedent. The triples will further have the same context, which combines R1 and R2.

ML-based Anaphoric Ambiguity Detection Our earlier work [3] indicates that, for the task of anaphoric ambiguity detection, (feature-based) ML leads to better accuracy than language modeling and off-the-shelf NLP methods. For anaphoric ambiguity detection, we


RS
txt
1
Loader
2
Reader
Context + Pronoun + Antecedent
Context + Pronoun
ML-based Anaphoric Ambiguity Detection
3
Language Features Extraction
4
Features Emebeddings Extraction
[LF$_1$, LF$_2$, …, LF$_n$]
[FE$_1$, FE$_2$, …, FE$_m$]
ML$_{FE}$ ML$_{LF}$
5
Classifier
label
SpanBERT-based Anaphora Resolution
6
Encoder
[CLS] C [SEP] P
7
Resolver
SpanBERT
Antecedent
CSV
Output
libraries.txt
RS: Requirements Specification, LF: Language Features, FE: Features Embeddings

```
nltk==3.5
spacy==2.3.7
benepar==0.2.0
stanza==1.2
pydotplus==2.0.2
nimbusml==1.8.0
scikit-plot==0.3.7
imblearn==0.0
tensorflow==2.4.1
textblob==0.15.3
sentence-transformers==2.0.0
openpyxl==3.0.9
et_xmlfile==1.1.0
```

Figure 2: Overview of TAPHSIR Architecture.

employ an ensemble ML classifier that combines the results of a classifier trained over language features (ML_{LF}) and another trained over feature embeddings (ML_{FE}). For training and applying ML classifiers, we use Scikit-learn 0.24.1 [16]. This component takes as input a set of triples associated with one the pronoun occurrence from the previous step, and derives as output a final label for that pronoun (ambiguous or unambiguous).

2.3 Language Features Extraction

This step extracts the different sets of learning features. In our work, we collected a set of 45 language features (LFs) from the NLP and RE literature. These features capture the characteristics of the relationship between the pronoun and its likely antecedent, e.g., both agree in gender or number. For extracting LFs, we use SpaCy 3.0.5 [7], NLTK 3.5 [11], Stanza 1.2 [18], and CoreNLP 4.2.2 [12]. The result of this step is a vector representing each input triple, where each entry in this vector is the result of computing an LF. For the example in Figure 1, we will generate three vectors representing the LFs of the pronoun "them" and each of its likely antecedents.

2.4 Extraction of Features Embeddings

This step extracts the feature embeddings (FEs) for each input triple. FEs are mathematical vectors that encapsulate the semantic and syntactic regularities of the sentence [9]. In our work, we extract 768 dimensional FEs from the BERT language model [2]. For that, we use the Transformers library, particularly the *bert-base-cased* model. Similar to the previous step, the output of this step is a vector representing each input triple. In a similar manner, this step results in three vectors for the example in Figure 1.

2.5 Classification

In this step, we pass the vector representation of each input triple to two pre-trained classifiers, namely ML_{LF} that is trained over LFs, and ML_{FE} trained over FEs. For each triple, the two classifiers independently predict a label as follows: *correct* (conversely, *incorrect*) indicating that the antecedent refers (conversely, does not refer) to the pronoun, or *inconclusive* when the anaphoric relation cannot be inferred. We then apply a set of rules on the predicted labels for the triples associated with one pronoun occurrence to conclude whether the pronoun is deemed ambiguous or unambiguous by each of the two classifiers. The rules, presented in the RE literature [22], consider the prediction probabilities produced for each possible antecedent.

Finally, we combine in an ensemble manner the results of the two classifiers ML_{LF} and ML_{FE} to derive the final label for the pronoun (i.e., ambiguous or unambiguous). If the two classifiers agree on the label (e.g., both conclude that the pronoun is ambiguous), then this label will be the final one for that pronoun. Otherwise, the label with the highest prediction probability will be selected. This ensemble learning method yields a more accurate prediction.

SpanBERT-based Anaphora Resolution. Based on the empirical findings in our earlier work [3], we know that for the task of anaphora resolution, the SpanBERT language model [8] outperforms alternatives. Consequently, the resolution component in TAPHSIR uses a SpanBERT model that is fine-tuned on a curated dataset from requirements. The dataset will be discussed in the next section. We implement SpanBERT using the Transformers 4.6.1 library [21] provided by Hugging Face (https://huggingface.co/) and operated in PyTorch [15]. This model takes as input, from the triples generated in the first step, only the pronoun and the context in which it occurs (i.e., disregards the likely antecedents). As SpanBERT is originally trained to extract text spans, SpanBERT in our work predicts as output the likely antecedent for the pronoun from its context.

2.6 Encoder

To be able to use SpanBERT model, the input pair of context and pronoun has to be encoded into the same format as the training data that BERT has been trained on. To do so, the input tuple is passed on to BERT's tokenizer which adds two special tokens: *[CLS]* to represent the classification output and *[SEP]* to separate the context from the pronoun occurrence. The token *[SEP]* informs BERT about which pronoun occurrence to analyze in the given context.

2.7 Resolver

In this step, we pass on the encoded input to the fine-tuned SpanBERT model and have the model predict the text span which likely represents the antecedent of the pronoun. SpanBERT can predict multiple such text spans with different probabilities indicating the likelihood of being the right antecedent. If an antecedent is predicted with a high probability (greater than 0.9), then we consider this as the resolution result for the pronoun.

2.8 Output

Given an input RS, the output of our tool is a csv file listing all pronoun occurrences in the input, and for each occurrence, providing the predicted label (ambiguous or unambiguous) and the most probable antecedent.

3 EVALUATION

In this section, we evaluate how accurately TAPHSIR can detect unacknowledged cases of anaphoric ambiguity and bring them to the attention of the requirements engineer.

3.1 Dataset Description

In this section, we use the curated dataset *DAMIR* (standing for Dataset for Anaphoric Ambiguity In Requirements) [3]. We curated this dataset with the help of two third-party annotators who underwent half-day training on ambiguity in requirements. We collected 22 industrial requirements specifications covering eight domains including satellite communications, medicine, aerospace, security, digitization, automotive, railway, and defence.

We preprocessed this collection and prepared the list of triples (a context, a pronoun occurrence and a possible antecedent) as explained in Section 2. The possible antecedents for a pronoun include all of the noun phrases preceding that pronoun [13]. The annotators then examined each pronoun occurrence and its possible antecedent considering the context in which they occur, and assigned a label *correct*, *incorrect*, or *inconclusive* with the same indications as explained in Section 2. We then post-processed the annotations and grouped them per pronoun occurrence as follows. We mark a pronoun as ambiguous in two cases: (i) if at least one annotator acknowledges the ambiguity of this pronoun by labeling one or more associated triples as *inconclusive*; or (ii) if the same triple associated with this pronoun receives different labels from the two annotators (e.g., *correct* versus *incorrect*). The former case implies acknowledged ambiguity, and the latter implies unacknowledged ambiguity.

As a result, *DAMIR* dataset contains a total of 737 pronoun occurrences that are analyzed for anaphoric ambiguity. About 46% of these pronouns (342/737) are deemed ambiguous by the annotators. Out of the ambiguous pronouns, we identified ≈87% with unacknowledged ambiguity, i.e., the annotators assumed that the pronoun is unambiguous yet had two different interpretations for that pronoun.

3.2 Results and Analysis

To assess how TAPHSIR performs in detecting unacknowledged ambiguity, we run TAPHSIR (depicted in Figure 2) on *DAMIR* dataset. TAPHSIR applies the an ensemble ML classifier for detecting ambiguity and SpanBERT for resolving anaphora as discussed in Section 2. On *DAMIR* dataset, TAPHSIR detects ambiguous cases with a perfect recall of 100% with a precision of ≈60%, while recommends automated resolution with an accuracy of ≈96% [3]. The perfect recall implies that TAPHSIR detects all unacknowledged ambiguous cases that were not explicitly marked by the human annotators as ambiguous.

The precision value indicates that the requirements engineer will invest some manual effort filtering out false positives, i.e., falsely detected ambiguous requirements. In the context of ambiguity in RE, recall is often favored over precision [1]. Achieving 100% recall ensures that all requirements suffering from all potentially ambiguous requirements will be brought to the attention of the engineers and further discussed at an early stage.

In a practical scenario where requirements engineers review requirements under time pressure, only the requirements that are found problematic by at least one engineer would be thoroughly discussed. The engineers might not discuss those requirements which they could confidently interpret unaware of having multiple inconsistent interpretations. In conclusion, we believe that TAPHSIR has a potential in practice since it perfectly detects also those requirements with unacknowledged ambiguity which would go otherwise unnoticed during manual inspection sessions. That said, a user study is required to assess the practical usefulness of the tool.

4 CONCLUSION

We presented TAPHSIR – a tool for detecting anaphoric ambiguity and resolving anaphora in natural-language requirements. Our current implementation reflects our findings in a multi-solution study [3]. TAPHSIR combines solutions based on machine learning and language models. We further evaluated how well TAPHSIR can detect unacknowledged ambiguity cases, i.e., the situation where different individuals perceive a requirement as unambiguous but, in reality, interpret the requirement differently. Our results show that TAPHSIR detects all ambiguous requirements (i.e., recall = 100%) including unacknowledged cases.

In future, we plan to do a user study to assess how useful is TAPHSIR in practice. Another topic for investigation is to use TAPHSIR as a bottom layer in a broader application in analyzing requirements, e.g., using the resolution results in an extracting domain model form a requirements specification.

ACKNOWLEDGMENTS

This work was funded by Luxembourg's National Research Fund (FNR) under the grant BRIDGES18/IS/12632261 and NSERC of Canada under the Discovery and Discovery Accelerator programs. We are grateful to the research and development team at QRA Corp. for valuable insights and assistance.

REFERENCES

[1] Daniel Berry. 2017. Evaluation of Tools for Hairy Requirements and Software Engineering Tasks. In *Proceedings of the 25th IEEE International Requirements Engineering Conference Workshops (REW'17)*. https://doi.org/10.1109/rew.2017.25

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. (2018). https://doi.org/10.48550/arXiv.1810.04805

[3] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated Handling of Anaphoric Ambiguity: A multi-solution Study. In *2022 IEEE/ACM 44th International Conference on Software Engineering*. https://doi.org/10.1145/3510003.3510157

[4] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C Briand. 2021. Using domain-specific corpora for improved handling of ambiguity in requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. https://doi.org/10.1109/ICSE43902.2021.00133

[5] Alessio Ferrari and Andrea Esuli. 2019. An NLP approach for cross-domain ambiguity detection in requirements engineering. *Automated Software Engineering* 26, 3 (2019). https://doi.org/10.1007/s10515-019-00261-7

[6] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. 2018. Detecting requirements defects with NLP patterns: An industrial experience in the railway domain. *Empirical Software Engineering* 23, 6 (2018). https://doi.org/10.1007/s10664-018-9596-7

[7] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. https://doi.org/10.5281/zenodo.1212303

[8] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics* 8 (2020). https://doi.org/10.1162/tacl_a_00300

[9] Dan Jurafsky and James H. Martin. 2020. *Speech and Language Processing* (3rd ed.). https://web.stanford.edu/~jurafsky/slp3/(visited 2021-06-04).

[10] Pohl Klaus and Rupp Chris. 2011. *Requirements Engineering Fundamentals* (1st ed.). Rocky Nook.

[11] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*.

[12] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. https://doi.org/10.3115/v1/p14-5010

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Neural Information Processing Systems Conference*.

[14] Ruslan Mitkov. 2014. *Anaphora resolution*. Routledge.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[17] Klaus Pohl. 2010. *Requirements Engineering* (1st ed.). Springer.

[18] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

[19] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. 2017. Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain. In *Proceedings of the 23rd Working Conference on Requirements Engineering: Foundation for Software Quality*. https://doi.org/10.1007/978-3-319-54045-0_24

[20] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace.

[21] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

[22] Hui Yang, Anne de Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. 2011. Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering* 16, 3 (2011). https://doi.org/10.1007/s00766-011-0119-y