Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments

Fouad Nasser A Al Omran
School of Computer Science
University of Adelaide
Adelaide, South Australia
Email: fouadnassera.alomran@student.adelaide.edu.au

Christoph Treude
School of Computer Science
University of Adelaide
Adelaide, South Australia
Email: christoph.treude@adelaide.edu.au

***Abstract*—To uncover interesting and actionable information from natural language documents authored by software developers, many researchers rely on "out-of-the-box" NLP libraries. However, software artifacts written in natural language are different from other textual documents due to the technical language used. In this paper, we first analyze the state of the art through a systematic literature review in which we find that only a small minority of papers justify their choice of an NLP library. We then report on a series of experiments in which we applied four state-of-the-art NLP libraries to publicly available software artifacts from three different sources. Our results show low agreement between different libraries (only between 60% and 71% of tokens were assigned the same part-of-speech tag by all four libraries) as well as differences in accuracy depending on source: For example, spaCy achieved the best accuracy on Stack Overflow data with nearly 90% of tokens tagged correctly, while it was clearly outperformed by Google's SyntaxNet when parsing GitHub ReadMe files. Our work implies that researchers should make an informed decision about the particular NLP library they choose and that customizations to libraries might be necessary to achieve good results when analyzing software artifacts written in natural language.**

***Keywords*—Natural language processing, NLP libraries, Part-of-Speech tagging, Software documentation**

I. Introduction and Motivation

Software developers author a wide variety of documents written in natural language, ranging from commit messages and source code comments to documentation and questions or answers on Stack Overflow. To automatically make sense of these natural language artifacts, industry and academia have developed techniques that rely on natural language processing (NLP): For example, Petrosyan et al. [1] developed a tool to automatically discover tutorial sections that explain the usage of an API type. Their approach relies on the Stanford NLP toolkit [2] for text classification. Another example of recent software engineering research relying on an NLP library is INTRUDER [3], a tool to produce multithreading tests that help detect atomicity violations. The authors used the open source NLP library OpenNLP to validate their work. In these examples, as in many other cases, software engineering researchers rely on existing NLP libraries, such as the Stanford NLP suite [2], Google's SyntaxNet [4], and NLTK [5]. These NLP libraries have been shown to achieve an accuracy of around 97% when applied to corpuses such as the Wall Street Journal, Treebank Union, and CoNLL '09 [6].

However, software artifacts written in natural language are different: Their language is technical and often contains references to code elements that a natural language parser trained on a publication such as the Wall Street Journal will be unfamiliar with. In addition, natural language text written by software developers may not obey all grammatical rules, e.g., API documentation might feature sentences that are grammatically incomplete (e.g., *"Returns the next page"*) and posts on Stack Overflow might not have been authored by a native speaker. Despite these challenges, many of the techniques that have been proposed to process software artifacts written in natural language rely on "out-of-the-box" NLP libraries, often without customizations or even a justification as to why a certain NLP library was chosen.

In this paper, we first present the results of a systematic literature review in which we analyzed 1,350 software engineering conference papers from the last five years in terms of whether they used NLP, what NLP library they used, and how this choice was justified. We found that out of 232 papers that mentioned "natural language", only 33 mentioned the specific library used. Moreover, the selection of the particular NLP library was justified in only two of these papers.

To help researchers and industry choose the appropriate library for analyzing software artifacts written in natural language, we then present the results of a series of experiments in which we applied four state-of-the-art NLP libraries to publicly available software artifacts from three different sources. We found that based on a total of 2,357 paragraphs analyzed, only 23,716 (91%) tokens were identified identically between all four libraries. The remaining 2,291 tokens were identified as separate tokens by some libraries whereas other libraries grouped them together. For example, some libraries split *"C++"* into three different tokens (*"C"*, *"+"*, *"+"*) while other libraries consider *"C++"* as a single token. Furthermore, based on a total of 26,006 tokens analyzed, only 16,607 (64%) of the tokens were given the same part-of-speech tags by all four libraries. In the remaining 9,399 cases, different libraries assigned different part-of-speech tags to the same token or they disagreed on the tokenization.

Based on manually annotating a subset of 1,116 randomly sampled tokens, we found that the spaCy library had the best performance on software artifacts from Stack Overflow and the Java API documentation while Google's SyntaxNet worked best on text from GitHub. The best performance was reached by spaCy on text from Stack Overflow (90% correct) while the worst performance came from SyntaxNet when we applied it to natural language text from the Java API Documentation (only 75% correct).

These results indicate that choosing an appropriate NLP library to analyze software artifacts written in natural language is indeed a critical step: The accuracy of state-of-the-art libraries differs greatly when applied to these artifacts (from 75% to 90% in our experiments) and even steps that could be considered as relatively straightforward, such as splitting a sentence into its tokens, become challenging when software artifacts are used as input.

In summary, in this paper we contribute:

- A systematic literature review considering 1,350 recent software engineering conference papers to investigate their use of NLP libraries. The data used for the systematic literature review is available for download and inspection in our online appendix.[1]
- Experiments with a total of 2,357 paragraphs and four state-of-the-art NLP libraries to analyze the extent to which the choice of library impacts the result of natural language processing.
- A manual annotation of 1,116 tokens with the correct part-of-speech tag and a comparison of the four NLP libraries based on this data. The corresponding data is available for download and inspection in our online appendix.[2]
- A tool to automate the comparison of the output of four NLP libraries based on a given corpus of natural language documents. The tool is available for download and inspection in our online appendix.[3]

The remainder of this paper is structured as follows: We present background on the different NLP libraries used in our investigation in Section II before detailing the methodology and results of our systematic literature review in Section III. Section IV introduces the tool that we developed to facilitate the comparison of the output of different NLP libraries—NLPCOMPARE—before we present the methodology and results of our experiments in Section V. Section VI describes our manual annotation and the results from comparing the NLP libraries based on this annotation. We discuss our findings in Section VII and threats to validity in Section VIII, and the work is concluded in Section IX.

II. BACKGROUND

Researchers can choose from a wide variety of NLP libraries to make sense of natural language artifacts. While it is common for researchers to rely on publicly available NLP libraries, some researchers develop their own tooling for specific tasks. For example, Allamanis et al. [7] developed a customized system called Haggis for mining code idioms and in our own previous work, we added customizations to the Stanford NLP library to improve the accuracy of parsing natural language text authored by software developers [8], [9]. In this work, we aim to identify how the choice of using a particular publicly available NLP library could impact the results of any research that makes use of an NLP library. To keep our workload manageable, we selected a subset of four libraries for our work: Google's SyntaxNet, the Stanford CoreNLP Suite, the NLTK Python library, and spaCy. We introduce each library and the reasons why we chose to include it in our work in the following paragraphs.

A. Google's SyntaxNet

SyntaxNet is an open-source neural network framework implemented in TensorFlow [4]. We selected it to be part of this work since it is the most recent addition to the set of publicly available NLP libraries, and based on the claim that it is the "world's most accurate Open Source parser" [4].

B. Stanford CoreNLP Suite

The Stanford CoreNLP Suite [2] is an integrated framework with the goal to facilitate the application of linguistic analysis tools to a piece of text. We chose to include it as part of our investigation because it is the most commonly used library in software engineering research concerned with natural language text. Stanford CoreNLP can be used as an integrated toolkit with a wide range of grammatical analysis tools and support for a number of major (human) languages [10]. In addition, it has a number of interfaces (wrappers) that are available for various major modern programming languages. This feature afforded us the freedom to select a programming language to interact with the CoreNLP API [10]. Moreover, the availability of resources and the number of users who have tried to enhance it were part of the reasons for considering Stanford CoreNLP as a library in this research.

C. NLTK Python Library

NLTK [5] is a Python library for analyzing human language data. Similar to the Stanford CoreNLP library, NLTK provides many wrappers for different programming languages and comes with many resources. We included it as part of this work since it is written in Python (unlike the Java-based Stanford CoreNLP Suite), and since it was trained on a wide range of corpora and lexical resources.

D. spaCy

The final library included in our work is spaCy [11], a library for advanced natural language processing written in Python and Cython. We decided to include spaCy in our work since it is primarily aimed at commercial applications and since our previous work (an extension of TaskNav [12]) showed that spaCy is substantially faster than many other libraries. As part of this work, we intend to compare the performance of all selected libraries in terms of their accuracy when parsing natural language text written by software developers.

[1]URL: http://tinyurl.com/Systematic-Literature-Review
[2]URL: http://tinyurl.com/NLPCOMPARE-Manual-Annotation
[3]URL: https://github.com/fouadrh5/NLPCOMPARE

III. Literature Review

As a first step of our investigation into the potential impact of choosing an NLP library for software engineering research and to identify what libraries researchers commonly use and how they justify their choice, we conducted a systematic literature review. In this section, we present the methodology of this review followed by its results.

A. Methodology

We focused our systematic literature review on the following top conferences in software engineering: The International Conference on Software Engineering (ICSE), the International Conference on Software Maintenance and Evolution (ICSME), the International Conference on Automated Software Engineering (ASE), the International Symposium on the Foundations of Software Engineering (FSE), and the International Conference on Mining Software Repositories (MSR). We limited the analysis to papers from the last five years at the time of the review, which led to the 2012–2016 timespan for most conferences, except for ICSME for which we analyzed papers published from 2011–2015 since the proceedings for 2016 were not available yet at the time of the review. This sampling method resulted in a total of 1,350 conference papers.

Since we were only interested in aspects of these papers with respect to natural language processing, we further limited the sample to papers that contained the phrase "natural language".[4] This phrase was derived from initial experimentation with different phrases and keywords.

For all 232 (17%) papers that contained the phrase, the first author read the paper, aiming to answer the following questions:

1) Did the authors employ natural language processing as part of their work?
2) What—if any—NLP library did they use?
3) Did they justify their choice of an NLP library?
4) If so, how was the choice justified?

While such details were usually covered in the methodology section, we also explicitly looked for pointers in the Threats to Validity sections of these papers, based on the possibility that authors might have used that section to explain why they decided to use one NLP library over another one.

B. Findings

Table I shows the number of papers that specified using a particular NLP library. Out of a total of 232 papers that mentioned "natural language", only 33 mentioned the specific library used. Of those, the majority use the Stanford CoreNLP library, while the other libraries are mostly used in a single paper. Out of the ones that we selected for our comparison, Google's SyntaxNet and spaCy have not been used at all in these recent software engineering conference papers. This is unsurprising since both libraries are fairly new. There is no significant difference between papers published in different conferences.

In terms of justifying their choice of NLP library, we found not a single paper that included a thorough justification. Two papers mentioned specific features of the library that they used—this can possibly be seen as a justification in the widest sense of the word possible: Zhai et al. [13] mentioned the following in relation to their use of the Stanford library: *"we leverage the state-of-the-art Stanford Parser with domain specific tags"*. In a similar way, Nguyen et al. [14] offered some details about the LangPipe library that they were using: *"we use LangPipe, a natural language processing tool which supports sentence splitting"*. While more detailed than other papers, both of these justifications are not sufficient to understand the thought process of the authors in choosing one library over others. Our detailed analysis of each paper is included in our online appendix.[5]

Given the lack of justification revealed by our literature review, the next step of our investigation is the comparison of the four selected libraries in terms of their impact. After all, if all libraries produce the same results when analyzing software development artifacts, no justification would be necessary for choosing one library over another. In the following, we first describe the tool we developed to facilitate the comparison.

IV. NLPCompare Tool Implementation

To study whether different NLP libraries parse the same textual content differently, we needed an infrastructure to easily compare the different libraries we chose to focus our investigation on (see Section II). In particular, we required tool support to give the same input to different libraries, capture the results at different levels (in particular tokenization and part-of-speech tagging), and compare the results between pairs of libraries and across all libraries. We focused our comparison on tokenization and part-of-speech tagging since these two aspects form the base of many other NLP features, such as named entity recognition (NER) and dependency parsing. Comparing these advanced features will be part of our future work.

To address the requirements posed by our study, we have implemented a Python[6] tool called NLPCompare to unify and simplify the process of running each library. NLPCompare outputs the result of the comparison into a Microsoft Excel spreadsheet to facilitate further analysis and easy readability of the results. In addition, the tool provides basic statistics of the comparison to give users first insights into the outcome of the comparison of different libraries.

We modified the output of all libraries to be in the format *"token"/"part-of-speech tag"* using the part-of-speech tags specified by the Penn Treebank Project [15] as the set of acceptable part-of-speech tags. Some of the libraries under investigation output different formats of part-of-speech tags—in those cases, we opted for the Penn Treebank option, i.e., no manual mapping was necessary.

Comparing the output of different NLP libraries is not trivial since different overlapping parts of the analysis need to be considered. Let us use the sentence *"Returns the C++ variable"* as an example to illustrate these challenges. The results of different NLP libraries differ in a number of ways:

[4] We used the following keywords to find relevant paragraphs in the selected papers: "parser", "token", "part-of-speech", and "tagging".

[5] URL: http://tinyurl.com/Systematic-Literature-Review

[6] Python Version 2.7.12

TABLE I: Use of NLP libraries in recent software engineering conference papers

Venue	Papers	Has Keyword	Stanford	SyntaxNet	NLTK	spaCy	LangPipe	OpenNLP	FudanNLP & IctcasNLP	in-house	others
ICSE	466	68	7	0	0	0	0	0	0	0	3
ASE	250	40	4	0	0	0	0	0	0	0	2
FSE	299	48	3	0	1	0	0	1	0	1	2
MSR	148	41	1	0	0	0	1	0	1	0	2
ICSME	187	35	3	0	0	0	0	0	0	0	1
Sum	1,350	232	18	0	1	0	1	1	1	1	10

1) Tokenization: Stanford's CoreNLP tokenizes *"C++"* as *"C"*, *"+"*, and *"+"* while the other libraries treat *"C++"* as a single token.
2) General part-of-speech tagging: Stanford's CoreNLP mis-classifies *"Returns"* as a noun, while the other libraries correctly classify it as a verb. This difference can be explained by the fact that the example sentence is actually grammatically incomplete—it is missing a noun phrase such as *"This method"* in the beginning.
3) Specific part-of-speech tagging: While several libraries correctly classify *"Returns"* as a verb, there are slight differences: Google's SyntaxNet classifies the word as a verb in 3rd person, singular and present tense (VBZ) while spaCy simply tags it as a general verb (VB).

Differences in tokenization imply differences in part-of-speech tagging (since different tokens are being tagged), and a different general part-of-speech tag implies that the specific part-of-speech tag is different as well. We argue that this last difference between a verb in 3rd person, singular and present tense (VBZ) and a general verb (VB) is not as big as the one between noun and verb, and therefore NLPCOMPARE considers these differences separately. In particular, we compare separately the general part-of-speech tag POS_g, represented by the first two letters of the part-of-speech tag (e.g., a noun, a proper noun, a plural noun, and a proper noun in plural form all share the same prefix: NN), and the specific part-of-speech tag POS_s.

V. EXPERIMENTS

To investigate the impact of choosing a particular NLP library and to help researchers and industry choose the appropriate library for analyzing software artifacts written in natural language, we have conducted a series of experiments in which we applied four NLP libraries to publicly available software artifacts from three different sources. We focused our analysis on artifacts related to the Java programming language in order to be able to compare results across different sources. We considered the following sources:

- Stack Overflow: We included Stack Overflow because of its size [16]. As of February 2017, Stack Overflow contained over 1.2 million questions tagged with "java", and the entire community of programmers active on Stack Overflow contained about 6.6 million programmers. The number of different people posting on the website has the potential to make natural language processing of the text particularly challenging since each will have their own writing style.
- GitHub ReadMe files: GitHub is home to open source projects written in 316 different programming languages, with Java being the second most commonly used language based on the number of opened pull requests (almost 800,000) [17]. Many GitHub projects contain ReadMe files which GitHub suggests to include in a project by default. Similar to Stack Overflow, these ReadMe files are authored by a wide variety of developers and are written in a variety of styles.
- Java API documentation: Compared to Stack Overflow and GitHub ReadMe files, the Java API documentation is well-structured and contributed to only by a small number of individuals. On the other hand, as the name suggests, API documentation contains the API specification which results in a large number of method and field definitions that would not be found in regular English text.

We detail the sampling methodology for each source in the next section followed by the findings.

A. Methodology

To reduce bias in our sample of textual content to be parsed by the four NLP libraries, we chose to randomly select paragraphs from each of the three sources. We discuss the sampling method for each source below. For all sources, we limited our analysis to artifacts related to Java and we removed code blocks—but not in-line code elements—from the text before using it as input for the NLP libraries.

1) Stack Overflow Random Sampling: We wrote two scripts that download random questions and answers through the Stack Overflow API. These scripts use a random number generator to select Stack Overflow question IDs lower than the highest ID (at the time of our sampling, the highest ID was 39,994,057). If the randomly selected question contains the "java" tag, one paragraph that is not a code block (i.e., not surrounded by the HTML tags `pre` and `code`) from the question text (title and body) is randomly selected and used as input to NLPCOMPARE. We repeated this sampling procedure until we had selected 200 paragraphs from Java-related Stack Overflow questions. We then used a similar approach to select 200 paragraphs from Java-related answers on Stack Overflow, resulting in a total of 400 paragraphs of Stack Overflow data to be passed to NLPCOMPARE.

2) GitHub ReadMe Random Sampling: Similar to the sampling for Stack Overflow, we also wrote a script to facilitate the random sampling of ReadMe files from Java-related GitHub projects. This script will retrieve random GitHub projects using

their ID. Since our focus is on Java, we have only considered projects written in this programming language as part of the sample. No other filters, such as popularity, time, and size, were applied.

If the randomly selected project contains a ReadMe file in the default location, we include the text of the ReadMe file in our sample. Since the GitHub API does not provide functionality to retrieve only the text of a file without markdown, we further removed GitHub markdown and any code blocks (i.e., paragraphs surrounded by ```) from the content. We sampled 20 GitHub projects, resulting in 547 paragraphs of text from the corresponding ReadMe files.

3) Java API Documentation Random Sampling: For the Java API documentation, we did not automate the sampling method. We randomly sampled 20 classes from the official Java API website [18] and considered all paragraphs that were not code blocks from the corresponding 20 documents, resulting in 1,410 paragraphs. Since there is no API to follow an approach for retrieving random paragraphs similar to the one used for the other two sources, we have sampled a higher number of paragraphs for Java API documentation.

B. Findings

Tables II and III as well as Figure 2 summarize our findings of applying NLPCOMPARE to the sampled text content from the three sources. Table III shows the number and percentage of identical tokens, tokens and general part-of-speech tags POS_g, and tokens and specific part-of-speech tags POS_s across all four NLP libraries for each source. We will describe and discuss the results in the following sections. The data used for the analysis below is available for download and inspection in our online appendix.[7]

1) Stack Overflow: The first part of Table II shows the results for analyzing the Stack Overflow sample with each of the four selected NLP libraries. Each row of the table represents one pairwise comparison of two of the libraries, and for each comparison, the table shows the number of tokens produced by the two libraries. For example, for the first comparison between Stanford's CoreNLP and Google's SyntaxNet, Stanford's CoreNLP produced a larger number of tokens (8,743) compared to SyntaxNet (8,485). As the fourth column shows, 8,137 of these tokens were identical between the two libraries, accounting for 94% of all tokens (i.e., number of identical tokens divided by the average number of tokens produced by the two libraries). Out of the identical tokens, 7,507 were assigned the same general part-of-speech tag POS_g, accounting for 87% of all tokens, and 7,322 tokens were assigned the same specific part-of-speech tag POS_s, accounting for 85% of all tokens. Recall from Section IV that we consider two tokens to have identical POS_g tags if the general part-of-speech is identical (e.g., noun, verb, adjective) whereas for POS_s tags to be considered identical, the part-of-speech tags have to be an exact match. For example, if one library tags a particular token as a noun (part-of-speech tag NN) and another tags the same token as a proper noun (part-of-speech tag NNP), the tags would be considered identical POS_g tags, but different POS_s tags. We make this distinction to account for the fact that identifying the specific part-of-speech POS_s is less important for parsing a sentence structure than the general part-of-speech POS_g.

a) Tokenization comparison: After applying NLPCOMPARE to our Stack Overflow sample, the agreement as shown in Table III among all libraries in terms of number of tokens is 7,899 identical tokens (92%). In more detail as represented in the first part of Table II, we found that different libraries tokenized the content differently. The total number of tokens based on the 400 paragraphs ranges from 8,485 to 8,743, and the overlap of tokens between two libraries is around 95%. An example of a disagreement between libraries regarding the tokenization of software-specific phrases is the phrase *"try-catch"*. Three of the libraries (Google's SyntaxNet, Stanford's CoreNLP, and NLTK) treat this phrase as a single token whereas spaCy turns it into three tokens: *"try"*, *"-"*, and *"catch"*.

b) Part-of-Speech Tagging comparison (General POS_g): As shown in Table III, comparing all libraries in terms of assigning the general part-of-speech tag POS_g for the Stack Overflow source has revealed 76% of identical assignment of POS_g. Specifically, we observed the highest agreement regarding POS_g between two libraries in the case of Google's SyntaxNet vs. spaCy (90%). All of the pairwise comparisons resulted in a value in the 80–90% range. The pair with the lowest agreement resulted from comparing the output of Stanford's CoreNLP and NLTK (81%, i.e., almost every fifth token was assigned a different general part-of-speech tag between these two libraries). Again, many of the differences can be traced back to vocabulary that is used in a software engineering context. For example, the word *"programmatically"* was correctly tagged as an adverb (RB) by three of the libraries (Google's SyntaxNet, spaCy, and NLTK), whereas Stanford's CoreNLP mis-classified it as a noun (NN). Another example is based on the sentence *"My program is going in the infinite loop"* where the token *"infinite"* was tagged correctly as an adjective (JJ) by three libraries (Google's SyntaxNet, Stanford's CoreNLP, and NLTK) while spaCy saw it as a noun (NN). We speculate that such problems are caused by libraries being unfamiliar with phrases such as *"infinite loop"* that are actually common in textual content specific to the software development domain.

c) Part-of-Speech Tagging comparison (Specific POS_s): Running NLPCOMPARE over this source using the specific set of tags POS_s resulted in a lower percentage of similarity of 71% as shown in Table III. In terms of pairwise comparison, the agreement was as low as 76% for Stanford vs. NLTK, i.e., almost every fourth token was assigned a different specific part-of-speech tag by these two libraries. The highest agreement was reached in the comparison of SyntaxNet vs. spaCy with a value of 88%. An example for a different specific part-of-speech tag comes from the sentence *"When I run my project it instantly crashes"* for which Stanford's CoreNLP and NLTK tagged the word *"run"* as a verb in its base form (VB) while Google's SyntaxNet and spaCy tagged it as a verb in past participle (VBN). The correct form would have been a verb in non-3rd person singular present (VBP), which was not assigned by any of the four libraries.

2) GitHub ReadMe files: The second part of Table II shows the results for the pairwise comparison of libraries when we

[7] URLs: http://tinyurl.com/NLPCOMPARE-NLPLibraries-Pairs, http://tinyurl.com/NLPCOMPARE-NLPLibraries-Pairs2

applied the libraries to the content of the sampled GitHub ReadMe files. While the overall results for tokenization are slightly better than for Stack Overflow data, the agreement in terms of part-of-speech tags is generally lower. The following paragraphs describe and discuss the results in detail.

a) Tokenization comparison: Looking at Table III for the overall performance from applying all libraries to the GitHub ReadMe files, we can see that the tokenization resulted in a higher percentage (94%) compared to the other two sources (Stack Overflow: 92%, Java API documentation: 89%). The number of tokens identified by the different libraries differs again, this time from 6,167 tokens (Google's SyntaxNet) to 6,289 tokens (Stanford's CoreNLP). The overlap between pairs of NLP libraries is above 95% for all pairs, with SyntaxNet and NLTK reaching the highest agreement (98%). An example of a disagreement is given by the URL *"http://releases.era7.com.s3.amazonaws.com"*, which was treated as a single token by Stanford's CoreNLP and spaCy while Google's SyntaxNet and NLTK produced four different tokens: *"http"*, *":"*, *"//"*, and *"releases.era7.com.s3.amazonaws.com"*.

b) Part-of-Speech Tagging comparison (General POS_g): For the general part-of-speech tag POS_g when comparing all libraries using GitHub data, Table III shows that the similarity percentage is around 71%. For the detailed comparison of general part-of-speech tags POS_g, the agreement between libraries was slightly lower for GitHub ReadMe files than for Stack Overflow data, ranging from 78% (between spaCy and NLTK) to 89% (between SyntaxNet and spaCy). An example disagreement was the treatment of the word *"generated"* in the sentence *"Read the content of generated indexes"*. It was tagged as a verb, past participle (VBN) by three libraries (Google's SyntaxNet, spaCy, and NLTK) while Stanford's CoreNLP tagged it as an adjective (JJ).

c) Part-of-Speech Tagging comparison (Specific POS_s): Applying our tool to compare the specific part-of-speech tags POS_s between all libraries has generated 62% agreement between them. More details are shown in the second part of Table II where the ratio of identical POS_s when comparing the result of parsing GitHub ReadMe files was as low as 69% in one case (spaCy vs. NLTK), i.e., these two libraries only agreed on just over two out of three part-of-speech tags. Values for other library pairs are slightly higher, in the 70–85% range. An example of a phrase where different libraries assigned different specific part-of-speech tags is *"Run-time"* which was tagged as a proper noun (NNP) by Stanford, as an adjective (JJ) by SyntaxNet and NLTK, and as a basic noun (NN) by spaCy. While the difference between noun and proper noun only affects the specific part-of-speech tags POS_s, the difference between adjective and noun also affects the general part-of-speech tag POS_g.

3) Java API Documentation: Table III and Figure 2 show that the percentage of identical tokens and part-of-speech tags is lower when applying the four libraries to Java API documentation compared to the other two sources. The last part of Table II shows the details of the comparison, and the following paragraphs discuss the details.

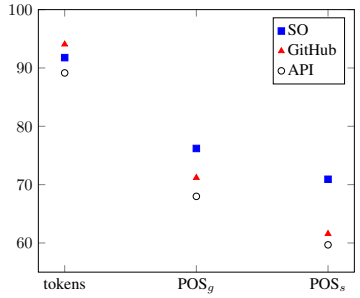


Fig. 1: Ratio of identical results across all libraries, in %

a) Tokenization comparison: When comparing the tokens generated by all libraries from Java API Documentation, only 89% of the tokens were split identically across all four libraries. In addition, the pairwise comparison of tokenization between Google's SyntaxNet and spaCy yielded the lowest similarity in all our experiments (91%). Many of the problems that the libraries had with tokenization could be traced back to API elements, which naturally occur frequently in API documentation. For example, the method name *"BorderUIResource.LineBorderUIResource"* was split into *"BorderUIResource"*, *"."*, and *"LineBorderUIResource"* by SyntaxNet and NLTK, while Stanford and spaCy treated it as a single token.

b) Part-of-Speech Tagging comparison (General POS_g): Similar to the tokenization results, the part-of-speech tagging also yielded the lowest agreement for the Java API documentation among all sources (only 68% agreement for the general part-of-speech tags POS_g). The last part of Table II shows that the highest agreement was between Google's SyntaxNet and spaCy with 83%, while Stanford and NLTK reached only 76% agreement. An example disagreement from the Java API documentation affects the word *"extended"* in the sentence *"Point on the extended line"*, which was tagged as a verb (VB) by Stanford and NLTK and as adjective (JJ) by SyntaxNet and spaCy.

c) Part-of-Speech Tagging comparison (Specific POS_s): Executing NLPCOMPARE over the Java API Documentation resulted in the lowest percentage of 60% identical specific part-of-speech tags POS_s among all sources, i.e., only about 3 out of 5 tokens are given the same tag. More details are represented in the last part of Table II: the highest similarity between two libraries is between SyntaxNet and spaCy with around 79%, and the lowest result is produced when comparing NLTK with either Stanford or spaCy (around 69%).

VI. MANUAL ANNOTATION

After establishing that different NLP libraries produce different results when they are applied to software artifacts written in natural language, our next step is to investigate which of the libraries achieves the best result. To that end, we manually annotated a sample of sentences with the correct token splitting and part-of-speech tags and compared the results of each library with this gold standard. The details of the methodology used for sampling sentences and for annotating them as well as the results of our analysis are discussed in the following subsections.

TABLE II: Degree of overlap between the results from all libraries

	Comparison	tokens for first library	tokens for second library	identical tokens	identical tokens %	identical token/POS$_g$	identical token/POS$_g$ %	identical token/POS$_s$	identical token/POS$_s$ %
Stack Overflow	Stanford vs. SyntaxNet	8,743	8,485	8,137	94.46	7,507	87.15	7,322	85.00
	Stanford vs. spaCy	8,743	8,589	8,312	95.92	7,743	89.35	7,548	87.10
	Stanford vs. NLTK	8,743	8,614	8,454	97.41	7,062	81.37	6,605	76.11
	SyntaxNet vs. spaCy	8,485	8,589	8,016	93.90	7,678	89.94	7,511	87.98
	SyntaxNet vs. NLTK	8,485	8,614	8,281	96.86	6,876	80.43	6,441	75.34
	spaCy vs. NLTK	8,589	8,614	8,326	96.80	7,035	81.79	6,585	76.56
GitHub ReadMe	Stanford vs. SyntaxNet	6,289	6,167	5,996	96.27	5,271	84.63	4,998	80.25
	Stanford vs. spaCy	6,289	6,278	6,034	96.03	5,332	84.86	5,100	81.16
	Stanford vs. NLTK	6,289	6,232	6,133	97.96	5,017	80.14	4,395	70.20
	SyntaxNet vs. spaCy	6,167	6,278	5,910	94.98	5,512	88.58	5,307	85.29
	SyntaxNet vs. NLTK	6,167	6,232	6,093	98.28	4,901	79.05	4,399	70.96
	spaCy vs. NLTK	6,278	6,232	6,040	96.56	4,885	78.10	4,313	68.95
Java API Doc.	Stanford vs. SyntaxNet	11,675	10,896	10,488	92.93	9,103	80.66	8,564	75.88
	Stanford vs. spaCy	11,675	11,015	10,611	93.53	9,390	82.77	8,832	77.85
	Stanford vs. NLTK	11,675	11,044	10,788	94.97	8,711	76.68	7,889	69.45
	SyntaxNet vs. spaCy	10,896	11,015	10,008	91.35	9,171	83.71	8,697	79.38
	SyntaxNet vs. NLTK	10,896	11,044	10,735	97.86	8,734	79.62	7,950	72.47
	spaCy vs. NLTK	11,015	11,044	10,311	93.49	8,602	77.99	7,713	69.93

TABLE III: Summary of identical tokens, tokens/POS$_g$, and tokens/POS$_s$ for all libraries

Source	average tokens	identical tokens	identical tokens %	identical token/POS$_g$	identical token/POS$_g$ %	identical token/POS$_s$	identical token/POS$_s$ %
Stack Overflow	8,608	7,899	91.77	6,559	76.20	6,105	70.92
GitHub	6,242	5,870	94.05	4,443	71.18	3,843	61.57
Java API	11,158	9,947	89.15	7,586	67.99	6,659	59.68

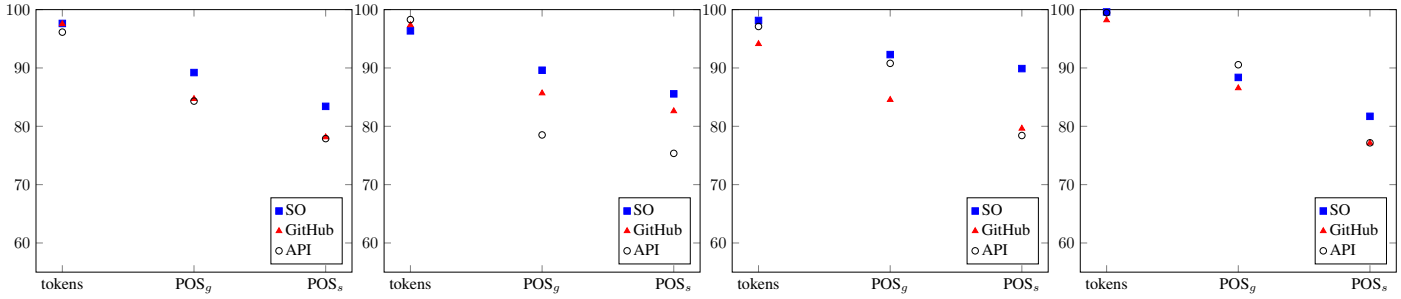
TABLE IV: Accuracy based on comparing the manual annotation with the output of the four libraries

	Comparison	tokens for manual annotation	tokens for library	identical tokens	identical tokens %	identical token/POS$_g$	identical token/POS$_g$ %	identical token/POS$_s$	identical token/POS$_s$ %
Stack Overflow	Manual vs. Stanford	375	385	371	97.63	339	89.21	317	83.42
	Manual vs. SyntaxNet	375	366	357	96.36	332	89.61	317	85.56
	Manual vs. spaCy	375	377	369	98.14	347	**92.29**	338	**89.89**
	Manual vs. NLTK	375	374	373	**99.60**	331	88.38	306	81.71
GitHub ReadMe	Manual vs. Stanford	361	371	357	97.54	310	84.70	286	78.14
	Manual vs. SyntaxNet	361	358	350	97.36	308	85.67	297	**82.61**
	Manual vs. spaCy	361	370	344	94.12	309	84.54	291	79.62
	Manual vs. NLTK	361	360	354	**98.20**	312	**86.55**	278	77.12
Java API Doc.	Manual vs. Stanford	380	398	374	96.14	328	84.32	303	77.89
	Manual vs. SyntaxNet	380	379	373	98.29	298	78.52	286	75.36
	Manual vs. spaCy	380	380	369	97.11	345	**90.79**	298	**78.42**
	Manual vs. NLTK	380	382	379	**99.48**	345	90.55	294	77.17

A. Methodology

To keep our workload manageable, we annotated a random sample of the sentences that were used in the experiments described in the previous section. We calculated the necessary sample size so that our conclusions would generalize to the entire data used in the previous experiments with a confidence level of 95% and a confidence interval of 5 [19]. We sampled tokens separately for each of the three sources (Stack Overflow, GitHub, and Java API documentation) by randomly sampling complete sentences until the number of tokens was equal to or larger than the required sample size. Since the number of tokens generated by each library is different, we chose the number of tokens generated by Stanford's CoreNLP as baseline since it produced the largest number of tokens in all our experiments. In the end, we annotated 375 tokens from Stack Overflow, 361 from GitHub ReadMe files, and 380 from the Java API documentation.

The authors of this paper manually annotated all 1,116 tokens with the correct part-of-speech tag. In some cases, the tokenization was also corrected, i.e., separate tokens were merged or single tokens were separated into different tokens. To verify the manual annotation, we asked a native speaker



(a) Stanford (b) SyntaxNet (c) spaCy (d) NLTK

Fig. 2: Comparing the manual annotation with the output of the four libraries, in %

to independently annotate the sampled tokens from Stack Overflow. There were 11 disagreements among the part-of-speech tags for all 375 tokens (less than 3%), which increases our confidence in the accuracy of the manual annotation. Most of the disagreements were between NN and NNP (i.e., deciding whether a code element is a noun or a proper noun) and between VBD and VBN (i.e., deciding between past tense and past participle in incomplete sentences). Only 4 disagreements (i.e., affecting about 1% of all tokens) indicated a different general part-of-speech tag POS$_s$.

In the following, we discuss the details of analyzing the manual annotation results for each NLP library.

B. Findings

Table IV and Figure 3 summarize our findings of applying NLPCOMPARE to the manual annotation that we performed over the sampled text. We will describe and discuss the results in the following sections.

1) Stack Overflow: The first part of Table IV depicts the information we have generated by running NLPCOMPARE over the Stack Overflow data. In general, Stack Overflow has the highest ratio of correctly identified tokens, POS$_g$ tags, and POS$_s$ tags compared to the Java API Documentation and GitHub ReadMe files.

a) Manual Annotation Tokenization comparison: NLTK achieved the highest agreement in tokenization with our manual annotation (373 of 375 tokens identical). As an example disagreement, we tokenized the Java command *"System.out"* as a single token which was mirrored in the tokenization produced by all libraries except for Stanford's CoreNLP which separated the command into three different tokens: *"System"*, *"."*, and *"out"*.

b) Manual Annotation Part-of-Speech Tagging comparison (General POS$_g$): For the general part-of-speech tags POS$_g$, the highest similarity with our annotation was achieved by spaCy with 92% while NLTK only achieved 88%. An example disagreement affected the word *"there"* in the sentence *"Is there a way to my goal by any algorithm"*. We tagged it as an existential there (EX) and both Google's SyntaxNet and spaCy agreed with our annotation, while Stanford's CoreNLP and NLTK disagreed and tagged it as an adverb (RB).

c) Manual Annotation Part-of-Speech Tagging comparison (Specific POS$_s$): Applying NLPCOMPARE to compare specific part-of-speech tags POS$_s$ between our manual annotation and all libraries also yielded the highest agreement in the case of spaCy (90%). An example disagreement comes from the sentence *"statement String Length"* where we tagged the word *"Length"* as a noun (NN) and both spaCy and NLTK agreed, but the other libraries have not and they tagged the word as a proper noun (NNP).

2) GitHub ReadMe Files: The second part of Table IV shows the result of running NLPCOMPARE to compare the manual annotation with the libraries' results in the case of GitHub ReadMe files. The following sections discuss the details.

a) Manual Annotation Tokenization comparison: The second part of Table IV shows that NLTK has the highest similarity (98%) in terms of identical tokens. Both Stanford's CoreNLP and Google's SyntaxNet have almost 97% agreement, and the lowest agreement is produced by spaCy with 94%. An example disagreement is the phrase *"compile-time"* which our manual annotation and most libraries treated as a single token, with the exception of spaCy, which produced the following three tokens: *"compile"*, *"-"*, and *"time"*.

b) Manual Annotation Part-of-Speech Tagging comparison (General POS$_g$): For the general part-of-speech tag POS$_g$, the highest agreement with our manual annotation is achieved by NLTK with 87% while the lowest agreement comes from both Stanford's CoreNLP and spaCy with around 84%. Google's SyntaxNet has better agreement with our manual annotation at 86%. For example, in the sentence *"with the various scanning solutions"* we tagged the word *"scanning"* as a noun (NN). However, only spaCy agreed, and the rest of the libraries have assigned different tags. In the case of Stanford, it tagged it as cardinal number (CD), and both Google SyntaxNet and NLTK libraries tagged it as present participle verb (VBG).

c) Manual Annotation Part-of-Speech Tagging comparison (Specific POS$_s$): Comparing our manual annotation with Google's SyntaxNet has produced the best result (83%) in terms of identical specific part-of-speech tags POS$_s$. The lowest value originated from NLTK with 77%. An example of a word that was tagged differently is *"scanning"* in this sentence *"the various scanning solutions"* which was tagged as a noun (NN) by Stanford's CoreNLP, spaCy, and our manual annotation. However, NLTK and Google's SyntaxNet tagged it as a verb (VB).

3) Java API Documentation: The last part of Table IV presents the result of comparing the manual annotation with the results from all NLP libraries over a sample from the Java API

Documentation. The details of each comparison are analyzed in the following subsections.

a) Manual Annotation Tokenization comparison: In terms of tokenization, we found that the highest agreement was achieved by NLTK (99%). Conversely, Stanford's CoreNLP has the lowest agreement with our manual annotation at 96%. An example disagreement is given by the tokenization of *"java.awt.Paint"* which was treated as a single token by the manual annotation, Google's SyntaxNet, and NLTK. Stanford's CoreNLP and spaCy produced five tokens as follows: *"java"*, *"."*, *"awt"*, *"."*, and *"Paint"*.

b) Manual Annotation Part-of-Speech Tagging comparison (General POS_g): NLTK and spaCy achieved around 90% agreement with our manual annotation in terms of the general part-of-speech tag POS_g. The lowest agreement resulted from Google's SyntaxNet at 79%. An example of a word that was tagged differently is *"color1"* in this sentence *"the first specified Point in user space color1"* which was tagged as a cardinal number (CD) by Google's SyntaxNet. The rest of the libraries including our manual annotation tagged it as noun (NN).

c) Manual Annotation Part-of-Speech Tagging comparison (Specific POS_s): Comparing our manual annotation with spaCy has produced the highest percentage (78%) among all NLP libraries in terms of the specific part-of-speech tags POS_s. An example of a disagreement is the word *"Shape"* in the sentence *"class provides a way to fill a Shape"* which Google's SyntaxNet tagged as a proper noun (NNP) in agreement with our manual annotation, while the other three libraries tagged it as a regular noun (NN).[8]

VII. Discussion

Our work raises two main issues. The first one is that many researchers apply NLP libraries to software artifacts written in natural language, but without justifying the choice of the particular NLP library they use. The second issue is that different NLP libraries actually generate different results, as shown in our experiments. This has implications on what library researchers and practitioners should choose when analyzing natural language artifacts created by software developers. We discuss both issues in more detail below.

A. Justifying the choice of NLP library

Despite the number of available NLP libraries, most researchers do not justify their choice of an NLP library. According to our systematic literature review of the top five software engineering conferences, 232 out of 1,350 recent papers contain the words "natural language", but only 33 mention the tools they use to deal with natural language. Out of those, only two mention some sort of justification, and many other papers apply Stanford's CoreNLP library without explicit reasoning. In our work with NLPCompare, we were able to show that the output of different libraries is not identical, and that the choice of an NLP library matters when they are applied to software engineering artifacts written in natural language. In addition, in most cases, Stanford's CoreNLP was outperformed by other libraries, and spaCy—which provided the best overall experience—was not mentioned in any recent software engineering conference paper that we included in our review.

B. Choosing an NLP library

In addition to showing that spaCy provides the best overall performance on the data that we used in our experiments, we also found that the choice of the best NLP library depends on the *task* and the *source*: For all three sources, NLTK achieved the highest agreement with our manual annotation in terms of tokenization. On the other hand, if the goal is accurate part-of-speech tagging, NLTK actually yielded the worst results among the four libraries for Stack Overflow and GitHub data. In other words, the best choice of an NLP library depends on which part of the NLP pipeline is going to be employed. In addition, while spaCy outperformed its competition on Stack Overflow and Java API documentation data, Google's SyntaxNet showed better performance for GitHub ReadMe files. Although we are not able to generalize this finding to other kinds of natural language artifacts authored by software developers, we provide evidence that the best library choice also depends on the nature of the text being analyzed.

The worst results were generally observed when analyzing Java API documentation, which confirms our initial assumption that the presence of code elements makes it particularly challenging for NLP libraries to analyze software artifacts written in natural language. In comparison, the NLP libraries were less impacted by the often informal language used in GitHub ReadMe files and on Stack Overflow. Going forward, the main challenge for researchers interested in improving the performance of NLP libraries on software artifacts will be the effective treatment of code elements.

By using NLPCompare, other researchers can build on these results and investigate the implications of choosing different NLP libraries in other contexts of software engineering data.

VIII. Threats to Validity

One threat to the validity of our results and an opportunity for future work lies in the fact that we used all four NLP libraries with their default settings[9] and without any specialized models. Also, the results are only reflecting the performance and accuracy of the current library versions which might change as the libraries are evolving. For some of the libraries, various models that have been trained by others on different data sets are available, and they might have achieved different results to the ones described here. Thus, our results only apply to the libraries' default setting and default models. However, since only a small minority of software engineering research papers that employ NLP libraries discuss specific settings of the library they use, we argue that this comparison of the default settings is a very important first step. We will investigate the impact of specific models and different settings in future work.

[8]Following the definition that proper nouns start with an upper-case letter, in our annotation, we generally tagged nouns as proper nouns if they were capitalized.

[9]Stanford parser version 3.7.0, Google SyntaxNet current version as of August 19th 2016, spaCy version 0.101.0, and NLTK version 3.2.1.

We expect that the best possible results will eventually be achieved by models trained specifically on natural language artifacts produced by software developers. Training and evaluating such a model is also part of our future work, and in this work we were able to provide evidence for its need. Our experiments showed that only 75–90% of all tokens were given the correct specific part-of-speech tag POS_s. Compared to the performance of these libraries on sources such as the Wall Street Journal (around 97%), these results have to be improved, and it seems reasonable to assume that a specifically trained model could outperform the results presented in this paper.

Another threat lies in the subjectivity of our manual annotation. However, we were able to recruit a native speaker who independently annotated part of our sample, and we found that the disagreement was less than 3% in terms of specific part-of-speech tags POS_s and less than 1% in terms of general part-of-speech tags POS_g. Due to the low disagreement between our annotation and hers as well as time constraints, we did not ask the native speaker to annotate the other sources.

One issue that we encountered when implementing NLP-COMPARE is that some libraries change the input (e.g., unifying all quotation marks or replacing parentheses with *"-LRB-"* and *"-RRB-"*, respectively). In some of these cases, we manually corrected the output to match the input again. While we double-checked all instances, it is possible that a small margin of error was introduced due to this issue.

IX. Related Work

There are a number of research efforts that have compared and evaluated NLP libraries for different purposes and sources. For example, Olney et al. [20] investigated the accuracy of nine NLP libraries including Stanford and spaCy in terms of part-of-speech tagging of 200 source code identifiers in open source Java programs. They concluded that part-of-speech taggers specifically built for source code, such as POSSE [21] and SWUM [22], were significantly more accurate than others. In our work, we compared the performance of the libraries on natural language artifacts written by software developers rather than source code identifiers, and we added the additional analysis of tokenization and pairwise comparison of libraries on top of part-of-speech tagging and a gold standard.

Tian and Lo [23] applied seven taggers that are part of four NLP libraries to 100 bug reports. The numbers for accuracy they found (between 83 and 91%) are similar to our findings. Our work complements theirs by adding three additional sources (Stack Overflow, GitHub ReadMe files, and Java API documentation), as well as using a different set of libraries and introducing an analysis of tokenization.

Choi et al. [24] provided a web-based tool to compare the dependency parser output for ten statistical dependency parsers (one of which is spaCy). Their comparison includes the accuracy and performance of each parser in terms of sentence length, dependency distance, projectivity, part-of-speech tags, dependency labels, and genre. We did not include dependencies in our analysis, but plan to expand our work to those in the future.

Arendse [25] recently provided an overview of the performance of NLP tools in the requirements engineering domain, with the goal to find defects and deviations in requirement documents. He measured the performance of NLP tools in terms of precision and recall of identifying specific undesired features, such as ambiguity. Our work is more widely applicable since it compares core NLP functionality.

Finally, in our own previous work [26], we investigated the challenges of analyzing software artifacts written in Portuguese. We analyzed 100 question titles from the Portuguese version of Stack Overflow with two Portuguese language tools and identified multiple problems which resulted in very few sentences being tagged completely correctly. Based on these results, we proposed heuristics to improve the analysis of natural language text produced by software developers in Portuguese. This work is similar in nature to the paper at hand, but here we use a much larger sample of data, a different language (English), and a more detailed analysis.

X. Conclusion and Future Work

Choosing the right NLP library is a critical step in any research project that chooses to employ natural language processing to analyze software artifacts written in natural language. To investigate the impact of choosing a particular NLP library, we first analyzed 1,350 recent software engineering conference papers and found that out of 232 papers that mentioned "natural language", only 33 mentioned the specific library used and only two of the papers provided rudimentary justification for the choice of library. Based on this finding, we conducted a series of experiments in which we applied four state-of-the-art NLP libraries (Google's SyntaxNet, Stanford's CoreNLP suite, the NLTK Python library, and spaCy) to publicly available software artifacts written in natural language from Stack Overflow, GitHub ReadMe files, and the Java API documentation. Based on a total of 2,357 paragraphs analyzed, we found that only 91% of the tokens were identified identically between all four libraries and only 64% of the tokens were given the exact same part-of-speech tag by all four libraries. In the remaining 36% of the cases, the libraries either disagreed on the part-of-speech tag or on the tokenization.

To help researchers and practitioners select the most appropriate NLP library for their task and source, we also compared the output of all four libraries on a sample of 1,116 tokens that we manually annotated with the correct part-of-speech tag. We found that which library performs best depends on the task (tokenization and part-of-speech tagging in our experiments) as well as on the source (in particular API documentation challenges some libraries more than others). Overall, we found that spaCy—which was not used at all in the papers that were part of our systematic literature review—achieved the most promising accuracy, but that there is much room for improvement.

This research is part of a bigger project in which we will employ NLP to automatically summarize software artifacts to help various stakeholders, such as newcomers [27], in software projects get high-level overviews of relevant information. Summarization is just one of many applications of NLP to software artifacts, and based on the experiments reported here, we are now able to make a more informed decision as to what NLP library to use to uncover interesting and actionable information from natural language documents authored by software developers.

ACKNOWLEDGEMENT

We thank Nancy Songtaweesin for independently annotating the sampled tokens from Stack Overflow to validate our manual annotation. We thank Saudi Aramco for supporting the first author's studies.

REFERENCES

[1] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining API types using text classification," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, 2015, pp. 869–879.

[2] "Stanford CoreNLP: a suite of core NLP tools," 2016. [Online]. Available: http://stanfordnlp.github.io/CoreNLP/

[3] M. Samak and M. K. Ramanathan, "Synthesizing tests for detecting atomicity violations," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 131–142.

[4] "Announcing SyntaxNet: The world's most accurate parser goes open source," 2016. [Online]. Available: https://research.googleblog.com/2016/05/announcing-syntaxnetworlds-most.html

[5] "Natural language toolkit NLTK 3.0 documentation," 2016. [Online]. Available: http://www.nltk.org

[6] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, "Globally normalized transition-based neural networks," *CoRR*, vol. abs/1603.06042, 2016.

[7] M. Allamanis and C. Sutton, "Mining idioms from source code," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 472–483.

[8] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 392–403.

[9] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 565–581, 2015.

[10] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60.

[11] "spacy.io," 2016. [Online]. Available: https://spacy.io

[12] C. Treude, M. Sicard, M. Klocke, and M. Robillard, "TaskNav: Task-based navigation of software documentation," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, 2015, pp. 649–652.

[13] J. Zhai, J. Huang, S. Ma, X. Zhang, L. Tan, J. Zhao, and F. Qin, "Automatic model generation from documentation for Java API functions," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 380–391.

[14] H. V. Nguyen, C. Kästner, and T. N. Nguyen, "Cross-language program slicing for dynamic web applications," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 369–380.

[15] E. Atwell, J. Hughes, and D. Souter, "Amalgam: automatic mapping among lexicogrammatical annotation models," in *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, J. Klavans, Ed. Association for Computational Linguistics, 1994, pp. 11–20.

[16] O. Barzilay, C. Treude, and A. Zagalsky, "Facilitating crowd sourced software engineering via Stack Overflow," in *Finding Source Code on the Web for Remix and Reuse*, S. E. Sim and R. E. Gallardo-Valencia, Eds. New York, NY: Springer New York, 2013, pp. 289–308.

[17] "The state of the Octoverse 2016," 2016. [Online]. Available: https://octoverse.github.com

[18] "Java platform, standard edition 7 API specification," 2016. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/

[19] "Sample size calculator," 2017. [Online]. Available: http://www.surveysystem.com/sscalc.htm

[20] W. Olney, E. Hill, C. Thurber, and B. Lemma, "Part of speech tagging Java method names," in *Proceedings of the International Conference on Software Maintenance and Evolution*, 2016, pp. 483–487.

[21] S. Gupta, S. Malik, L. Pollock, and K. Vijay-Shanker, "Part-of-speech tagging of program identifiers for improved text-based software engineering tools," in *Proceedings of the 21st International Conference on Program Comprehension*, 2013, pp. 3–12.

[22] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 43–52.

[23] Y. Tian and D. Lo, "A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 570–574.

[24] J. D. Choi, J. R. Tetreault, and A. Stent, "It depends: Dependency parser comparison using a web-based evaluation tool." in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 387–396.

[25] B. Arendse, "A thorough comparison of NLP tools for requirements quality improvement," Master's thesis, Utrecht University, 2016.

[26] C. Treude, C. A. Prolo, and F. Figueira Filho, "Challenges in analyzing software documentation in Portuguese," in *Proceedings of the 29th Brazilian Symposium on Software Engineering*, 2015, pp. 179–184.

[27] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 273–284.