SCALAR: A Part-of-speech Tagger for Identifiers

Christian D. Newman*, Brandon Scholten†, Sophia Testa†, Joshua A. C. Behler†,
Syreen Banabilah†, Michael L. Collard‡, Michael J. Decker§, Mohamed Wiem Mkaouer¶,
Marcos Zampieri‖, Eman Abdullah AlOmar** Reem Alsuhaibani†† Anthony Peruma ‡‡
Jonathan I. Maletic†

*Department of Software Engineering, Rochester Institute of Technology, Rochester, NY, USA, 0000-0002-8838-4074
†Department of Computer Science, Kent State University, Kent, OH, USA,
(bscholte, stesta4, jbehler1, sbanabil, jmaletic)@kent.edu
‡Department of Computer Science, University of Akron, Akron, OH, USA, collard@akron.edu
§Department of Software Engineering, Bowling Green State University, Bowling Green, OH, USA, mdecke@bgsu.edu
¶Department of Computer Science, University of Michigan Flint, Flint, MI, USA, mmkaouer@umich.edu
‖George Mason University School of Computing, Fairfax, VA, USA, mzampier@gmu.edu
**Stevens Institute of Technology, School of Engineering and Science, Hoboken, NJ, USA, ealomar@stevens.edu
††Prince Sultan University Department of Software Engineering, Riyadh, Saudi Arabia, rsuhaibani@psu.edu.sa
‡‡University of Hawaii at Manoa Department of Information and Computer Sciences, Honolulu, HI, USA, peruma@hawaii.edu

***Abstract*—The paper presents the Source Code Analysis and Lexical Annotation Runtime (SCALAR), a tool specialized for mapping (annotating) source code identifier names to their corresponding part-of-speech tag sequence (grammar pattern). SCALAR's internal model is trained using scikit-learn's GradientBoostingClassifier in conjunction with a manually-curated oracle of identifier names and their grammar patterns. This specializes the tagger to recognize the unique structure of the natural language used by developers to create all types of identifiers (e.g., function names, variable names etc.). SCALAR's output is compared with a previous version of the tagger, as well as a modern off-the-shelf part-of-speech tagger to show how it improves upon other taggers' output for annotating identifiers. The code is available on Github[1]**

***Index Terms*—Program comprehension, identifier naming, part-of-speech tagging, natural language processing, software maintenance, software evolution**

I. Introduction

The identifiers developers create represent a significant amount of the information other developers must use to understand related code. Given that identifiers represent, on average, 70% of the characters in a code base [1], and developers spend more time reading code than writing [2], [3], it is important for researchers to better understand of how identifiers convey information, and how they can be improved to increase developer reading efficiency. This problem is complicated by the fact that there are multiple comprehension styles [4], [5], and the influence of identifier naming at varying levels of experience is currently understudied, especially in education contexts [6], [7]. Further, while there are many studies that correlate the influence of identifiers (and various related characteristics) on comprehension [8]–[11], research has not found formal rules or unification among the outcomes of these studies due to the complexity of the problem, and the need for further research.

[1]https://github.com/SCANL/scanl_tagger

Thus, research is needed to improve identifier naming practices. Many approaches try to improve identifiers through predicting words that should appear in a name [12], [13] or analyzing/normalizing identifiers [14]–[23] to understand how well they fit within a coding context, or make them easier to process. However, a significant challenge in identifier naming research lies in measuring the semantics of identifier names, then using that information to critique/generate better names.

Measurement of identifier name semantics requires mapping between the terms and their meaning as an identifier (i.e., a sequence of terms). The ability to do this in a formal way allows us to recommend sequences of terms based on an understanding of the semantics they typically convey together. It also allows us to determine when the terms used in a sequence are inappropriate, since certain sequences are uncommon or even known to be anti-patterns. This is a challenge, as there are many ways to cluster identifiers using the terms used to construct them. Prior research shows that using grammar patterns to cluster/group identifiers with similar part-of-speech (PoS) sequences is an effective way to study how different types of identifiers convey meaning [24]–[26], and has the advantage that every term that is part of a natural language can be associated with a PoS tag. In addition, PoS tags are a known and well-supported means to formally define the function of a term within a sequence. Thus, we can group identifiers by grammar patterns (i.e., their PoS sequence) to measure the information they convey, and what patterns are most commonly used to convey different types of information. This is in contrast, and potentially complementary, to other approaches that cluster raw terms, or use vector representations.

Thus, we present SCALAR: A part-of-speech tagging approach specialized for source code identifiers. SCALAR is explicitly designed to support the generation of grammar pattern sequences to support future research and development of techniques that leverage grammar patterns. The goal of this

TABLE I
PART-OF-SPEECH CATEGORIES IN DATASET AND SUPPORTED BY SCALAR

Abbreviation	Expanded Form	Examples
N	noun	stack, function, language
DT	determiner	the, this, that, these, those, which
CJ	conjunction	and, for, nor, but, or, yet, so
P	preposition	behind, in front of, at, under, beside, above, beneath, despite
NPL	noun plural	strings, identifiers, classes
NM	noun modifier (**noun adjunct**, adjective)	**employee**Name, **token**Parser, dynamic
V	verb	run, execute, implement, develop
VM	verb modifier (adverb)	quickly, safely, eventually
PR	pronoun	she, he, her, him, it, we, us, they, them, I, me, you
D	digit	1, 2, 10, 4.12, 0xAF
PRE	preamble*	Gimp, GLEW, GL, G

paper is to show the effectiveness of SCALAR for generating grammar patterns. SCALAR can be used in the future to improve techniques to analyze, recommend, and critique identifiers, by generating grammar patterns that can be used to understand identifier name meanings [24]–[27]. SCALAR is still very much in development as we apply it in our research and modify it to assist. However, it is a working tool that will be useful for others in identifier-oriented work.

II. Related Work

POSSE [19] and SWUM [28], and SCANL tagger [29] are part-of-speech taggers created specifically to be run on software identifiers; they are trained to deal with the specialized context in which identifiers appear. Both POSSE and SWUM take advantage of static analysis to provide annotations. For example, they will look at the return type of a function to determine whether the word *set* is a noun or a verb. Additionally, they are both aware of common naming structures in identifier names. For example, methods are more likely to contain a verb in certain positions within their name (e.g., at the beginning) [19], [28]. They leverage this information to help determine what POS to assign different words. Olney et al. [30] compared taggers for accuracy on 200+ identifiers, but only on Java method names. They found that SWUM and POSSE were the most accurate taggers for source code at the time of publication. Newman et al. [24] compared the same taggers but on a larger dataset (1,335 identifiers) and five identifier categories: function, class, attribute, parameter, and declaration statement. They found that SWUM was the most accurate overall, with an average accuracy around 59.4% at the identifier level. Later, Newman et al. created a new tagger and compared with SWUM, POSSE, and Stanford [31], finding that their new tagger exceeded the others' performance metrics on identifier names [29].

TABLE II
EXAMPLES OF GRAMMAR PATTERNS

Identifier Example	Grammar Pattern
action to index map	N P NM N
as binary	P N
time for each line	N P DT N
server and port	N CJ N
open if empty	V CJ NM
adjust to camera	V P N

III. Methodology

The core of SCALAR is a GradientBoostingClassifier that is trained using the combination of two manually-curated data sets of identifiers and their corresponding grammar patterns. The first data set is called the General Grammar Dataset (GGD). The second dataset is called the Closed Grammar Dataset (CGD). The first dataset is used to train the first iteration of SCALAR [29], while the second is created to improve on the original: It underperformed on closed syntactic categories such as preposition, conjunction, and determiner.

The GGD is made up of 1,335 identifiers from 5 contexts: function, declaration, attribute, parameter, and class name. It represents a 95 and 6 sample from a dataset of identifiers from 20 open source systems [29]. The CGD is made up of 1,275 identifiers, representing a 95 and 5 sample from a dataset of 30 systems validated similarly to prior work [29]. The difference between these is that the CGD contains a higher population of closed-category words, such as prepositions, conjunctions, etc. We used srcML [32] to do all data collection and filtering.

We combine these datasets to construct the Training Dataset (TD). This dataset contains a total of 1,335 + 1,275 = 2,610 identifiers. This translates to 7,173 rows, each row containing one word from the identifiers in our dataset. As stated, we train SCALAR using scikit-learn's GradientBoostingClassifier algorithm. The training is split set into train (70%, 5021 words), and test (30%, 2152 words) using a stratified, random sample. We stratified on the ground-truth PoS annotation for each word. We used stratified k-fold cross validation with k=10 for training to help improve the generality of SCALAR.

A number of features are used to help train the model. One of the major differences between the prior Ensemble Tagger [29] and SCALAR is that it significantly reduces its reliance on external taggers, with NLTK [33] being the only other tagger whose output SCALAR is trained on. Instead, this tagger relies on word-embedding features and lexical features inspired by our prior work on grammar patterns [24], [25]. This makes SCALAR much faster than its predecessor and equally, or more, accurate across the range of PoS categories. Due to space limitations, we will not go into detail about every new feature, but the strongest features in terms of their importance metric are as follows:

1) **NLTK_POS**. We use the NLTK part of speech tagger as a feature due to its speed, and it provides an off-the-shelf tagger perspective to SCALAR; it roots SCALAR

in traditional PoS tagging, allowing it to focus on specializing the tags to the unique context of code.

2) **Preposition Embeddings**. We collect a list of common prepositions (we reference word sources above the lists in the code[2]), translate them into word embeddings, then use an average, normalized vector of those embeddings to determine whether a given word is close (in terms of angles between vectors) to the general concept of a preposition. We do the same with **nouns** and **verbs** to create average noun/verb word embedding vectors.
3) **Ratio of word position**. The position of a word within an identifier can provide valuable information about its role. For example, words at the end of an identifier tend to be nouns (i.e., head nouns); the word at the beginning of a function identifier tends to be a verb. This feature represents the ratio between given word's position and the length of the identifier that it is part of; it gives us an idea of how 'far' into an identifier a given word is.

Our tagset is specialized for identifiers found in code. This tagset (shown in Table I) is first discussed in Newman et al.'s original work on Grammar Patterns [24]. In this paper, Newman et al. shows that identifiers follow unique grammatical rules that are rarely in most natural human language text. Thus, they argue that specialized taggers are necessary for identifiers. Most of these tags are available and known to general PoS tagging approaches. However, there are two tags in our set that we must discuss, since their usage within identifiers is part of what sets the natural language in code apart from other natural language contexts, like newspapers. These are Noun Modifiers (NM) [19], [24], [28] and Preambles (PRE) [24], [28]. A Preamble is an abbreviation which does one of the following:

1) Namespaces an identifier without augmenting the reader's understanding of its behavior (e.g., XML in XML_Reader is not a preamble)
2) Provides language-specific metadata about an identifier (e.g., identifies pointers or member variables)
3) Highlights an identifier's type. When a preamble is highlighting an identifier's type, the type's inclusion must not add any new information to the identifier name.

We give examples of each preamble type in the list. An example of (1) can be found in the GIMP and GLEW open-source projects, where GIMP and G_ are namespace preambles to many variables. To discuss (2), we use Hungarian notation [34]. Hungarian notation is when developers, for example, put *p_* in front of pointer variables or m_ in front of variables that are members of a class; any Hungarian notation in an identifier is considered a preamble. As an example of (3), given the declaration *float* fPtr*, 'f' in 'fPtr' does not add any information about the identifier's role within the system, but reminds the developer that it has a type 'float'; **this is a preamble**. However, given an identifier *char* sPtr*, 's' informs the developer that this is a c-string as opposed to a pointer to some other type of character array; 's' is **not** considered a preamble under this definition above. Intuitively, the reason for identifying preambles in an identifier is because they do not provide any information with respect to the identifier's role within the system's domain. Instead, they provide one of the types of information above.

[2]https://github.com/SCANL/scanl_tagger/blob/master/feature_generator.py

TABLE III
TEST SET METRICS PER TAGGER. EACH TAGGER WAS RUN ON THE SAME TEST SET, AND METRICS WERE GATHERED FROM THEIR PER-WORD PERFORMANCE.

	Accuracy	Balanced Accuracy	Weighted Recall	Weighted Precision	Weighted F1	Performance (seconds)
SCALAR	**0.8216**	**0.9160**	**0.8216**	**0.8245**	**0.8220**	**249.05**
Ensemble	0.7124	0.8311	0.7124	0.7597	0.7235	1149.44
Flair	0.6087	0.7844	0.6087	0.7755	0.6497	807.03

Another tag to note in Table I is *noun modifier (NM)*, which is annotated on words that can be considered a pure adjective or noun-adjunct. A noun-adjunct is a word that is typically a noun but is being used as an adjective. An example of this is the word *bit* in the identifier *bitSet*. In this case, *bit* is a noun which describes the type of *set*, i.e., it is a set of bits. So we consider it a noun-adjunct. These are found in English (e.g., compound words), but generally not annotated as their own individual PoS tag. Prior work argues for the use of an individual tag for noun-adjuncts due to their ubiquity, and special role, in source code identifiers [19], [24], [28].

Our evaluation is performed at the level of words and not full identifiers, since annotating even one word incorrectly within an identifier makes the annotation for the entire identifier incorrect. Word-level analysis is more granular and still correlates with higher correctness over whole identifiers.

IV. EVALUATION

We perform a comparison of our PoS tagger against an off-the-shelf part of speech tagger called Flair [35], as well as the previous iteration of our tagger, the Ensemble Tagger, which was shown to be the most accurate tagger compared to an off-the-shelf tagger (Stanford [31]) and code-specialized taggers (SWUM [28], Posse [19]) in prior work [29].

In order to compare with Flair [35], we need to translate Flair's output into our tagset; the translation between SCALAR's tagset and Flairs can be found in our Git repo README[3] where we show how to convert Penn Treebank to our tagset. In mapping to Penn Treebank, some granularity is lost. For example, we map most verb variation forms to just verb. This decision is based on prior experience with how verb variations are used in code, and is explained in more detail in prior work [24]. In summary, reducing these variations to verb simplifies the task of comparing, and many of these variations have uses in code that cause them to behave as non-verbs. For example, *waitingList* has a verb (waiting), but it is being used as an noun modifier (describing the type of list). Refer to [24] for more on that issue. Note that the purpose of comparing to Flair is primarily to show that an off-the-shelf PoS tagger cannot be readily used to annotate

[3]https://github.com/SCANL/scanl_tagger

TABLE IV
CATEGORY-LEVEL METRICS FOR SCALAR BASED ON TEST SET PERFORMANCE

	N (Noun)			V (Verb):			NM (Noun Modifier):			D (Digit):			P (Preposition):		
	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair
Precision:	0.798	0.7247	**0.8854**	**0.7413**	0.5473	0.5678	0.8075	**0.8687**	0.2516	0.957	0.9032	**0.989**	**0.929**	0.6129	0.8452
Recall:	**0.8258**	0.8209	0.5293	**0.8514**	0.6322	0.689	**0.762**	0.6326	0.6169	**0.957**	0.9438	0.8036	**0.9351**	0.6934	0.8506
F1 Score:	**0.8116**	0.7698	0.6625	**0.7926**	0.5867	0.6226	**0.7841**	0.7321	0.3574	**0.957**	0.9231	0.8867	**0.932**	0.6507	0.8479
	VM (Verb Modifier):			PRE (Preamble)			DT (Determiner):			NPL (Noun Plural):			CJ (Conjunction):		
	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair	SCALAR	Ensemble	Flair
Precision:	0.75	0.2917	**0.9**	**0.701**	0.1856	0	**0.9697**	0.3434	0.4444	0.9204	0.8761	**0.9646**	**0.8235**	0.5294	0.625
Recall:	**0.8182**	0.3333	0.2609	**0.7816**	0.6	0	**0.8421**	0.7907	0.7857	**0.8525**	0.8319	0.7899	**0.9333**	0.5625	0.8333
F1 Score:	**0.7826**	0.3111	0.4045	**0.7391**	0.2835	0	**0.9014**	0.4789	0.5677	**0.8851**	0.8534	0.8685	**0.875**	0.5455	0.7143

with the specialized grammatical structure of identifiers. It is not designed to recognize this specialized structure and, as a result, it significantly under-performs versus its relatively high accuracy on normal PoS tagging tasks. That is, even if we use Flair's tagset, it will still under-perform. This is clear from the performance on the NM category in Table IV. Instead of recognizing identifiers like bitSet as a noun-adjunct and a noun (NM N), Flair recognizes it as two nouns (N N). This does not correctly identify the relationship between these words. For more information, and examples, refer to Newman et al. [24], [25].

Compared with the Ensemble Tagger [29], its predecessor, SCALAR is somewhat better in terms of accuracy, precision, recall, and F1, but the true advantage that SCALAR has above its predecessor is *speed*. SCALAR is much faster, annotating all 2152 words in the test data set in 249.05 seconds, versus the 1149.44 seconds it took the Ensemble Tagger. That's 0.12 (249.05/2152) seconds per word versus 0.53 (1149.44/2152) seconds per word; SCALAR is 4.42 times faster while remaining more effective than its predecessor.

In terms of overall performance, SCALAR generally out-performs the other two taggers on identifiers in terms of our performance metrics at a macroscopic level (Table III), and on a per-category basis (Table IV).

V. THE APPLICATION

SCALAR runs as a Python Flask server using Waitress, which opens the tagger to a user-defined address and port and has the ability to handle simultaneous web requests from users. This allows SCALAR to be available for an internal group to use. HTTP requests are sent to SCALAR that contain the identifier to process. SCALAR is very customizable, and allows for user specified acceptable words and abbreviations. Thus any domain-specific terms that do not have a standard dictionary definition are manually flagged as valid words and reported with corrected PoS information.

SCALAR returns JSON output with a list containing each word found in an identifier. For each word, the output includes PoS information and an additional tag indicating if it is a standard dictionary word. Every time SCALAR encounters an identifier for the first time, it caches the results of the splitter and tagger. Every subsequent time SCALAR encounters the same identifier, it returns the cached information on the identifier. This caching considerably speeds up the splitting and tagging process, eliminating the need to process an identifier which has been previously tagged. This provides a significant performance increase for users who work in code bases where the same identifiers are reused frequently. Running the application on a couple systems we found that the average time for a result of an identifier for the first time is 133.2 milliseconds and after caching (second time) the average time for a result for the same identifier is 1.2 milliseconds.

Additionally, SCALAR saves the first time it encounters an identifier, the most recent encounter, and the number of times it encounters an identifier. This provides a log of identifiers and is useful for researchers to better understand identifier uses. The first and last encounters are saved as UNIX timestamps and displayed in the JSON output.

SCALAR is available for download as a Docker image. The Docker image automates the process of setting up an environment in which SCALAR can run by including all of the dependencies in a container. The Dockerfile included with SCALAR downloads the required Python packages, word embeddings, English word dictionary, and a list of allowable domain-specific words and abbreviations. Downloading this information from a separate source every time the docker image is built allows the lists to be easily updated by rebuilding the Docker image. Once these items are downloaded, the docker image automatically runs the commands to train the tagger and start the Python Flask server to receive requests. The port on which SCALAR listens for requests is mapped to port 8080 (a standard port for web traffic) on the machine hosting the docker container. As requests are sent to SCALAR, a JSON file containing the result cache is stored in a docker volume. Storing the cached output allows for data to be collected about the identifiers sent to the tagger during a specific time frame.

VI. CONCLUSION

This paper introduces SCALAR, a part-of-speech tagger for source code identifiers. It is trained using several features involving word embeddings and an external part-of-speech tagger output. It is designed to support the generation of grammar patterns [24]–[26] for the purpose of analyzing, critiquing, and improving identifier names. SCALAR is faster than its predecessor, has similar performance, and out-performs other PoS taggers on identifier names.

REFERENCES

[1] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, p. 261–282, Sep. 2006. [Online]. Available: https://doi.org/10.1007/s11219-006-9219-1

[2] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, 1989.

[3] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[4] A. von Mayrhauser and A. M. Vans, "Program understanding behavior during debugging of large scale software," in *Papers Presented at the Seventh Workshop on Empirical Studies of Programmers*, ser. ESP '97. New York, NY, USA: ACM, 1997, pp. 157–179. [Online]. Available: http://doi.acm.org/10.1145/266399.266414

[5] M. Fisher, A. Cox, and L. Zhao, "Using sex differences to link spatial cognition and program comprehension," in *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*, ser. ICSM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 289–298.

[6] V. van der Werf, A. Swidan, F. Hermans, M. Specht, and E. Aivaloglou, "Teachers' beliefs and practices on the naming of variables in introductory python programming courses," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, ser. ICSE-SEET '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 368–379. [Online]. Available: https://doi.org/10.1145/3639474.3640069

[7] E. L. Glassman, L. Fischer, J. Scott, and R. Miller, "Foobaz: Variable name feedback for student code at scale," *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:15810023

[8] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, and M. Beigl, "Descriptive compound identifier names improve source code comprehension," in *Proceedings of the 26th Conference on Program Comprehension*, ser. ICPC '18. New York, NY, USA: ACM, 2018, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/3196321.3196332

[9] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 217–227.

[10] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. IEEE, 2010, pp. 156–165.

[11] E. Avidan and D. G. Feitelson, "Effects of variable names on comprehension: An empirical study," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 55–65.

[12] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Suggesting accurate method and class names," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 38–49. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786849

[13] K. Liu, D. Kim, T. F. Bissyandé, T. Kim, K. Kim, A. Koyuncu, S. Kim, and Y. Le Traon, "Learning to spot and refactor inconsistent method names," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE 2019. New York, NY, USA: ACM, 2019.

[14] J. Zhang, S. Liu, L. Gong, H. Zhang, Z. Huang, and H. Jiang, "Beqain: An effective and efficient identifier normalization approach with bert and the question answering system," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2597–2620, 2023.

[15] N. Al Madi, "Namesake: A checker of lexical similarity in identifier names," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3560441

[16] V. Arnaoudova, L. M. Eshkevari, M. D. Penta, R. Oliveto, G. Antoniol, and Y.-G. Gueheneuc, "Repent: Analyzing the nature of identifier renamings," *IEEE Trans. Softw. Eng.*, vol. 40, no. 5, pp. 502–532, May 2014. [Online]. Available: https://doi.org/10.1109/TSE.2014.2312942

[17] E. W. Høst and B. M. Østvold, "Debugging method names," in *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, ser. Genoa. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 294–317. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03013-0_14

[18] D. Binkley, M. Hearn, and D. Lawrie, "Improving identifier informativeness using part of speech information," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 203–206. [Online]. Available: http://doi.acm.org/10.1145/1985441.1985471

[19] 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 *2013 21st International Conference on Program Comprehension (ICPC)*, May 2013, pp. 3–12.

[20] M. Alghamdi, S. Hayashi, T. Kobayashi, and C. Treude, "Characterising the knowledge about primitive variables in java code comments," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 460–470.

[21] Y. Osumi, N. Umekawa, H. Komata, and S. Hayashi, "Empirical study of co-renamed identifiers," in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, 2022, pp. 71–80.

[22] C. D. Newman, M. J. Decker, R. S. AlSuhaibani, A. Peruma, D. Kaushik, and E. Hill, "An empirical study of abbreviations and expansions in software artifacts," in *Proceedings of the 35th IEEE International Conference on Software Maintenance*. IEEE, 2019.

[23] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "An empirical study of identifier splitting techniques," *Empirical Softw. Engg.*, vol. 19, no. 6, pp. 1754–1780, Dec. 2014.

[24] C. D. Newman, R. S. AlSuhaibani, M. J. Decker, A. Peruma, D. Kaushik, M. W. Mkaouer, and E. Hill, "On the generation, structure, and semantics of grammar patterns in source code identifiers," *Journal of Systems and Software*, vol. 170, p. 110740, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220301680

[25] C. Newman, M. Decker, and R. Alsuhaibani, "Identifier name structure catalogue." [Online]. Available: https://github.com/SCANL/identifier_name_structure_catalogue.

[26] A. Peruma, E. Hu, J. Chen, E. A. AlOmar, M. W. Mkaouer, and C. D. Newman, "Using grammar patterns to interpret test method name evolution," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, 2021, pp. 335–346.

[27] A. Peruma and C. D. Newman, "Understanding digits in identifier names: An exploratory study," in *Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering*, ser. NLBSE '22. New York, NY, USA: Association for Computing Machinery, 2023, p. 9–16. [Online]. Available: https://doi.org/10.1145/3528588.3528657

[28] E. Hill, "Integrating natural language and program structure information to improve software search and exploration," Ph.D. dissertation, Newark, DE, USA, 2010, aAI3423409.

[29] C. D. Newman, M. J. Decker, R. S. Alsuhaibani, A. Peruma, M. W. Mkaouer, S. Mohapatra, T. Vishnoi, M. Zampieri, T. J. Sheldon, and E. Hill, "An ensemble approach for annotating source code identifiers with part-of-speech tags," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3506–3522, 2022.

[30] W. Olney, E. Hill, C. Thurber, and B. Lemma, "Part of speech tagging java method names," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct 2016, pp. 483–487.

[31] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, ser. EMNLP '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, pp. 63–70. [Online]. Available: https://doi.org/10.3115/1117794.1117802

[32] M. L. Collard and J. I. Maletic, "srcml 1.0: Explore, analyze, and manipulate source code," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct 2016, pp. 649–649.

[33] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[34] C. Simonyi and M. Heller, "The hungarian revolution," *BYTE*, vol. 16, no. 8, p. 131–ff., Aug. 1991.

[35] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *COLING 2018, 27th International Conference on Computational Linguistics*, 2018, pp. 1638–1649.