Modeling the Relationship Between Identifier Name and Behavior

Christian D. Newman*, Anthony Peruma*, Reem AlSuhaibani†
*Rochester Institute of Technology, Rochester, NY, USA
†Kent State University, Kent, OH, USA, Prince Sultan University, Riyadh, Saudi Arabia
cnewman@se.rit.edu, axp6201@rit.edu, ralsuhai@kent.edu

***Abstract*—This paper presents the features of a model that relates the natural language found in identifiers with program semantics. The model takes advantage of part of speech information and static-analysis-based program models to understand how different types of statically-derived semantics correlates with the natural language meaning of identifiers.**

***Index Terms*—Program Comprehension, Identifier Names, Static Analysis**

I. Introduction

Currently, developers spend a large amount of time comprehending code [1], [2]; 10 times the amount they spend writing it by some estimates [2]. One important aspect of comprehension is the naming of identifiers (e.g., class names, function names, parameter names); weak identifier names decrease productivity [3]–[7], normalizing identifier names helps both developers and research tools [8], [9], and many research projects aim to improve identifier naming [10]–[20].

Even though there are many approaches to measuring the quality of identifiers and recommending better identifier names, research has yet to discover a way to produce a model which fully captures the relationship between natural language tokens and program semantics. Understanding how natural language and program semantics relate to one another will enable automated generation of identifiers; make software development more accessible to different types of people (e.g., different types of learners/readers) at different stages of experience (e.g., identifiers made to help novices comprehend versus experts); and would be a strong step forward in understanding how developers model programs mentally.

There is a wide range of perspectives in literature on how to improve naming. Some use token frequency with different models (e.g., SMTs, neural networks) and some static analysis to predict tokens that should appear in an identifier name [10], [11], [13], [14]; others combine natural language (NL) and static analysis, using NL techniques to reveal the role of words which constitute identifiers. This role is then mapped to statically-defined program semantics [15], [18], [19], [21]–[24]; and some take a fully static-analysis-based approach to group identifiers by common semantic role based purely on programming language rules and no NL analysis [16], [17], [25]. Each of these perspectives approach a similar problem but from a different angle. In this work, we aim to extend the second and third perspectives by using part of speech (POS) tagging on groups of words present in identifiers and connecting this with static analysis-based models that group identifiers by their common semantics. The goal of this paper is to describe the basis of an extensible model combining NL and program semantics.

TABLE I: Example Features of the Proposed Model

Identifier Example	Grammar Pattern
1. GList* **tile_list_head** = NULL;	adjective adjective noun
2. GList* **tile_list_tail** = NULL;	adjective adjective noun
3. Gulong **max_tile_size** = 0;	adjective adjective noun
4. GimpWireMessage **msg**;	noun
5. **g_list_remove_link** (tile_list_head, list)	preamble noun verb noun
6. **g_list_last** (list)	preamble adjective noun
7. **g_assert** (tile_list_head != tile_list_tail);	preamble verb
S-Lexical Category	**Stereotype**
1. s-pronoun (**tile_list_head**)	N/A
2. s-pronoun (**tile_list_tail**)	N/A
3. s-adjective (**max_tile_size**)	N/A
4. proper s-noun (**msg**)	N/A
5. s-verb (**g_list_remove_link**)	Command
6. s-verb (**g_list_last**)	Getter
7. s-verb (**g_assert**)	Predicate

II. Modelling NL and Program semantics

The model discussed in this paper significantly extends and combines previous work by Binkley et al. [18], Newman et al. [16], AlSuhaibani et al. [17], and Dragan et al. [25]. Binkley studied patterns of part of speech tags in identifiers to help understand the structure of class identifier names; Newman and AlSuhaibani proposed a model that groups identifiers together by their declarations, and Dragan proposed a model for grouping methods by semantics. These four papers form part of the basis of the proposed work, which will model what the identifier can do (e.g., constraints placed on it by its declaration [16], [17], [25]) with what it says it does (i.e., by analyzing the terms used in a given identifier using part of speech [18]) and how it interacts with the rest off the program through function calls. The model is generated by looking for statistically-significant, empirically-derived, patterns between statically-captured program semantics and part of speech. It extends the works cited above by combining the models that underpin each work into a single, cohesive model which can be easily extended to include more features (e.g., we immediately extend it by including function call data). We give an example of the model features that we use in Tables I and II.

TABLE II: Example of Usage Patterns

tile_list_head usages
static GList * tile_list_head = NULL
list = tile_list_head
g_assert (tile_list_head != tile_list_tail)
tile_list_head = g_list_remove_link (tile_list_head, list)
while (tile_list_head)
if (!tile_list_head)
tile_list_head = tile_list_tail
tile_list_head = g_list_remove_link (tile_list_head, list)
tile_list_tail usages
static GList * tile_list_tail = NULL
if (list == tile_list_tail)
g_assert (tile_list_head != tile_list_tail)
tile_list_tail = g_list_last (g_list_concat (tile_list_tail, list))
g_hash_table_insert (tile_hash_table, tile, tile_list_tail)
tile_list_tail = g_list_last (tile_list_tail)
tile_list_head = tile_list_tail

A. Grammar Patterns

A grammar pattern is the set of part of speech tags that correspond to the individual terms that make up an identifier (i.e., identifiers can be made of one or more terms concatenated together). To assign a grammar pattern to an identifier, we must split [26] the identifier first and then run a part of speech tagger to obtain a part of speech tag for the terms that make up that identifier. In Table I, the Grammar Pattern (top right) segment gives the grammar pattern for the corresponding (**bolded**) identifier name in the Identifier Example segment. So the grammar pattern for tile_list_head is *adjective adjective noun*, (*tile-list* modifies/describes *head*) in terms of parts of speech, and the same for tile_list_tail. Here, the grammar pattern represents the natural language characteristics that our model aims to map into source code semantics.

B. S-Lexical Categories

On the bottom left of Table I, we provide the s-lexical category for the **bolded** identifiers [16], [17]. S-Lexical categories are a taxonomy of static-analysis-based categories that group identifiers based on their type declarations. They are able to categorize any identifier that has a type (i.e., function name, parameters, local/globals, etc). They group identifiers that have similar constraints on their semantics (i.e., due to their declaration) together. The categories in this taxonomy are named after natural language part of speech tags prepended with an *s-*. For example, *s-pronouns* are identifiers whose declaration indicates that they behave in a similar fashion as pointers in C/C++. Specifically, they may refer to different entities depending on context, much like pronouns in NL. We refer the reader to the original publication for a more formal definition. By grouping using both the s-lexical categories and grammar patterns, we can correlate different grammar patterns and their relationship to multiple identifiers' type-based constraints.

C. Method Stereotypes

On the bottom right, we provide the stereotype [25] label for function identifiers. Stereotypes classify functions based on the role they play in the context of a class. For example, *command* stereotyped functions are mutators that execute a complex change to the calling objects state. Stereotypes only work on functions, so other identifiers have no label. The s-lexical categories label function names at a high level (i.e., they are either s-verbs or s-adjectives). The stereotypes are much more granular; allowing us to break function name identifiers down to more specific categories and correlate grammar patterns with this information. As with the s-lexical categories, the goal here is to correlate common grammar patterns to different stereotypes in order to connect source code semantics (i.e., via stereotypes) with natural language (i.e., via grammar patterns).

D. Usage Patterns

Table II gives an example of usage patterns, which are function calls that a given identifier has participated in (e.g., as an argument or calling object). For this example, we have taken two identifiers and listed out every one of their usages from Gimp [27]. The S-lexical category attributed with both the *tile_list_head* and *tile_list_tail* identifiers is s-pronoun and the grammar pattern for each is *adjective adjective noun*. In terms of S-lexical category and grammar patterns, they are the exact same. If we compare the literal words in each identifier, they are the same up to the very last term in each (i.e., head and tail) and both these terms have the same POS tag (i.e., noun). If we look at their usages, they have two expressions in common (in **bold**); the rest differ, however. The *tile_list_tail* identifier is used in methods like append(), insert(), and last(). By contrast, *tile_list_head* is used in remove() and in looping structures. The point here is that the two identifiers are very similar because they are related; the S-lexical categories and grammar patterns reveal their relatedness. However, there are some important differences that we miss. Specifically, one is the head and the other is the tail. Considering usage patterns allow us to discriminate more effectively. We can use as many or as few usage patterns as required and can consider inter-procedural use cases using slicing [28], [29].

III. Conclusion

One problem we will apply this model to is identifier name appraisal; assigning a quality rating to a name. This will work by considering how well an identifier's grammar pattern, declarations, and usages (i.e., the features of our model) match up to other identifiers with similar characteristics. Some long term goals of this research are to 1) provide a method of suggesting names for developers, 2) generate names automatically for code generation tools, and 3) help create a theory of optimal naming; specifically, how to assign the best words to an identifier. The model's toolset will be built using srcML [30] and opened to the community to encourage and support more research in this area.

IV. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1850412.

REFERENCES

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

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

[3] 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*, ICPC '18, (New York, NY, USA), pp. 31–40, ACM, 2018.

[4] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pp. 3–12, June 2006.

[5] 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)*, pp. 217–227, Feb 2017.

[6] 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*, pp. 156–165, IEEE, 2010.

[7] A. A. Takang, P. A. Grubb, and R. D. Macredie, "The effects of comments and identifier names on program comprehensibility: an experimental investigation," *J. Prog. Lang.*, vol. 4, pp. 143–167, 1996.

[8] D. Binkley, D. Lawrie, and C. Morrell, "The need for software specific natural language techniques," *Empirical Softw. Engg.*, vol. 23, pp. 2398–2425, Aug. 2018.

[9] 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 and Evolution (ICSME)*, IEEE, 2019.

[10] 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*, ESEC/FSE 2015, (New York, NY, USA), pp. 38–49, ACM, 2015.

[11] K. Liu, D. Kim, T. F. Bissyand, 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*, ICSE 2019, (New York, NY, USA), ACM, 2019.

[12] E. W. Høst and B. M. Østvold, "Debugging method names," in *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, (Berlin, Heidelberg), pp. 294–317, Springer-Verlag, 2009.

[13] S. L. Abebe and P. Tonella, "Automated identifier completion and replacement," in *2013 17th European Conference on Software Maintenance and Reengineering*, pp. 263–272, March 2013.

[14] Y. Kashiwabara, Y. Onizuka, T. Ishio, Y. Hayase, T. Yamamoto, and K. Inoue, "Recommending verbs for rename method using association rule mining," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pp. 323–327, Feb 2014.

[15] 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, pp. 502–532, May 2014.

[16] C. D. Newman, R. S. AlSuhaibani, M. L. Collard, and J. I. Maletic, "Lexical categories for source code identifiers," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 228–239, Feb 2017.

[17] R. S. Alsuhaibani, C. D. Newman, M. L. Collard, and J. I. Maletic, "Heuristic-based part-of-speech tagging of source code identifiers and comments," in *2015 IEEE 5th Workshop on Mining Unstructured Data (MUD)*, pp. 1–6, Sep. 2015.

[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*, MSR '11, (New York, NY, USA), pp. 203–206, ACM, 2011.

[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)*, pp. 3–12, May 2013.

[20] S. L. Abebe and P. Tonella, "Automated identifier completion and replacement," in *2013 17th European Conference on Software Maintenance and Reengineering*, pp. 263–272, March 2013.

[21] V. Arnaoudova, M. Di Penta, G. Antoniol, and Y. Guhneuc, "A new family of software anti-patterns: Linguistic anti-patterns," in *2013 17th European Conference on Software Maintenance and Reengineering*, pp. 187–196, March 2013.

[22] A. Peruma, M. W. Mkaouer, M. J. Decker, and C. D. Newman, "Contextualizing rename decisions using refactorings and commit messages," in *Proceedings of the 19th IEEE International Working Conference on Source Code Analysis and Manipulation*, IEEE, 2019.

[23] A. Peruma, M. W. Mkaouer, M. J. Decker, and C. D. Newman, "An empirical investigation of how and why developers rename identifiers," in *International Workshop on Refactoring 2018*, 2018.

[24] E. Hill, L. Pollock, and K. Vijay-Shanker, "Improving source code search with natural language phrasal representations of method signatures," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 524–527, Nov 2011.

[25] N. Dragan, M. L. Collard, and J. I. Maletic, "Reverse engineering method stereotypes," in *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*, ICSM '06, (Washington, DC, USA), pp. 24–34, IEEE Computer Society, 2006.

[26] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "An empirical study of identifier splitting techniques," *Empirical Softw. Engg.*, vol. 19, pp. 1754–1780, Dec. 2014.

[27] S. Kimball, P. Mattis, and T. G. D. Team, "Gimp."

[28] H. Alomari, M. Collard, J. I. Maletic, N. Alhindawi, and O. Meqdadi, "srcslice: very efficient and scalable forward static slicing," *Journal of Software: Evolution and Process*, vol. 26, 11 2014.

[29] C. D. Newman, T. Sage, M. L. Collard, H. W. Alomari, and J. I. Maletic, "srcslice: A tool for efficient static forward slicing," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 621–624, May 2016.

[30] 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)*, pp. 649–649, Oct 2016.