Contextual Similarity Among Identifier Names: An Empirical Study

Remo Gresta[1], **Elder Cirilo**[1]

[1]Federal University of São João del-rei

Abstract. *Identifiers are one of the most important sources of domain information in software development. Therefore, it is recognized that the proper use of names directly impacts the code's comprehensibility, maintainability, and quality. Our goal in this work is to expand the current knowledge about names by considering not only their quality but also their contextual similarity. To achieve that, we extracted names of four large scale open-source projects written in Java. Then, we computed the semantic similarity between classes and their attributes/variables using Fasttext, an word embedding algorithm. As a result, we could observe that source code, in general, preserve an acceptable level of contextual similarity, developers avoid to use names out of the default dictionary (e.g., domain), and files with more changes and maintained by distinct contributors tend to have better a contextual similarity.*

1. Introduction

The source code readability plays an important role in software comprehension, especially when documentation is scarce or not available. Indeed, several researchers have shown the value of readable source code to enhance software quality and their relevance in evolution tasks. In particular, identifiers are the most prevalent entities in the source code. In any ordinary software, approximately 70% of its tokens are identifiers [Deissenboeck and Pizka 2006]. Therefore, an important element of software comprehension is to understand the underlying concepts embodied in the code by means of decoding identifier names [Avidan and Feitelson 2017].

The importance of meaningful identifier names was established in several papers. For example, Hofmeister et al. [Hofmeister et al. 2017] showed that full word identifiers may lead to better comprehension than identifiers composed of single letters. However, as pointed out by Deissenboeck et al. [Deissenboeck and Pizka 2006], a significant proportion of the source code vocabulary can be either acronyms, abbreviations or concatenation of terms that can not be easily identified and do not follow any naming convention. Butler et al. [Butler et al. 2009] also resort to study the connections between identifier names and bugs. They found statistically significant associations between names with pour quality and code quality issues reported by static analysis tool (e.g., FindBugs). As may be expected, names chosen by one developer can not also convey the expected meaning in a working context, impairing software comprehension. Although, Feitelson et al. [Feitelson et al. 2020] observed that experienced developers tend to use longer and good quality identifier names, there has been relatively little research investigating to what extent identifiers names co-existence (e.g., in files, classes or method body). It is, therefore, appealing to observe how names are chosen together in a specific context (e.g., whether they tend to be semantically similar or not), and whether the process of choosing meaningful names evolves.

Our goal in this work is to expand the current knowledge about identifier names by considering not only their quality in terms of length or compounded words, but taking into account their contextual similarity (semantic similarity among identifier names in the context of a class). To achieve that, we extracted names of four large scale open-source projects written in Java (i.e., Tomcat, Spring-boot, Jenkins, and Fastjson). Then, we computed the semantic similarity between class names and their attributes/variables using Fasttext, an word embedding algorithm.

As a result, we could observe that:

- developers tend to preserve an acceptable level of contextual similarity among names in the source code.
- developers tend to use existing words to name identifiers, that is, they usually avoid the use out of the dictionary (e.g., domain) words.
- the number of changes and distinct contributors (i.e., authors) tend to be associated with better contextual similarity.

2. Background and Related Work

We start this section by presenting the relevance of providing proper identifier names and how it might affect source code comprehensibility, maintainability, and quality. We also overview some related work and provide some background about word embedding and how it is able to give well suited semantic similarity scores to words in a working context.

2.1. Identifiers and Names

According to Deissenboeck et al. [Deissenboeck and Pizka 2006], approximately 2/3 of any regular source code is composed of identifiers. Therefore, identifiers are more than names to local variables, methods, attributes, and even classes; they are, alongside comments, the main sources of domain information. An identifier's name can be: a fully spelled word; the abbreviation of a word; or the combination of two or more words. Names might also involve words that in fact do not exist or even be single alphabetical characters. In general, a fully spelled word is more descriptive than a single character. Indeed, the proper use of names has been recognized as a major issue in software development. Lawrie at. al. [Lawrie et al. 2006], have shown that the use of full words names on identifiers assists developers on maintainability tasks. In their study, developers were able to better understand code snippets in contexts where full words were chosen as the name, instead of a single letter variant. Naturally, choosing low-quality or unrelated names end up resulting in a source code that is more difficult to maintain and that can be associated with bugs [Butler et al. 2010] [Li et al. 2018] [Kawamoto and Mizuno 2012].

2.2. Source Code as Word Embedding

Learning the intrinsic meaning of a word can be an easy task for a human, but it is not a so trivial task for computer machines. A solution is to represent words as vectors, or as word embeddings, as they are often called [Jurafsky and Martin 2000]. In word embeddings, words are mapped to real number vectors so that words with similar meanings are going to have similar vectors. Therefore, word embeddings aim to preserve/represent the contextual semantic similarities among words [Mikolov et al. 2013]. That is, the meaning of words can be established using words that are usually close to them.

From representatives word vectors we can calculate which we call the semantic similarity among words. The most common ways to realize that are: measuring the cosine of the angle among vectors; or by calculating the euclidean distance among vectors. The resulting value is going to be to 1 whether the meaning of the words is similar, and -1 whether they are not. There are popular models designed for generating word embeddings: word2vec [Mikolov et al. 2013], Global Vector for Word Representations (GLoVe) [Pennington et al. 2014] and Fasttext by Facebook [Mikolov et al. 2017]. In this work, we choose by Fasttext as a model to evaluate the semantic similarity among existing names in the source code of the subject projects.

3. A Method for Evaluating Contextual Similarity in Source Code

This section presents the method which we conducted for evaluating the semantic similarity among names in four large scale open-source projects written in Java. We consider that: object-oriented programming encourages developers to use naming conventions which make the source code more understandable and easier to read.

3.1. Extracting Names

We can consider as an identifier: variables, classes, methods, parameters, and attributes. In this work, we attained to study the semantic similarity among class names, attributes, and variables. We used Srcml[1] to extract the studied identifier's names. Srcml is a multi-language parsing tool that converts source code into srcML – an XML representation in which markup tags represent the elements of the language abstract syntax. We created a tool that read the generated XML files and extracts names from the `<name>` tags inside other tags mean class statement declaration, attribute statement declaration, and variable statement declaration.

3.2. Computing Contextual Similarity

To compute the semantic similarity among the extracted names in a context, we used the Gensim Library[2] – an open-source library focused on topic modeling, document indexing and similarity evaluation in large corpora. Gensim provides already pre-trained and high-quality word vectors learned from large data-bases. Considering that to calculate the semantic similarity among words they have to exist in the corpora, we choose by the corpus *Wikipedia 2017*, an 1 million words pre-trained model. We are assuming that the majority of words can be found in our chosen dictionary.

To evaluate the contextual similarity among names we designed two metrics:

- **Internal Similarity**: it means the semantic similarity among names present in a file. For each class in a file, we proceed by computing the similarity between its name and the name of all of its attributes and variables; and so, the median of the resulting similarities. The result is a score varying from -1 to 1, where: 1 means a high internal similarity; and -1 means a low Internal Similarity. For example, consider the class `ExtractResources` in Figure 1. As we can observe, the names `resourceName` (attribute - Line 2) and `resource` (variable - Line 8) are both well aligned with the class name `ExtractResource`. Indeed, the

[1] www.srcml.org

[2] radimrehurek.com/gensim/

```
public class ExtractResources extends DefaultTask {
    private final Map<String, String> properties = new HashMap<>();
    private List<String> resourceNames = new ArrayList<>();
    ...
    @TaskAction
    void extractResources() throws IOException {
        for (String resourceName : this.resourceNames) {
            ...
            String resource = FileCopyUtils
                   .copyToString(new InputStreamReader(resourceStream,
                                                        StandardCharsets.UTF_8));
            ...
        }
    }
}
```

```
public class MavenConsoleAnnotator extends
                   LineTransformationOutputStream.Delegating {
    private final Charset charset;
    ...
    @Override
    protected void eol(byte[] b, int len) throws IOException {
        Matcher m = MavenMojoNote.PATTERN.matcher(line);
        if (m.matches())
            new MavenMojoNote().encodeTo(out);
        ...
    }
}
```

Figure 1. Example of classes with high and low semantic similarities respectively

`ExtractResource` class has a high internal similarity, equals to 0.43658462. In contrast, the class `MavenConsoleAnnotator` (see Figure 1) has a low internal similarity, equals to -0.09242306. There is almost any similarity among names in this example. The names `charset` and `m` have little or no similarity to the name `MavenConsoleAnnotator`.

- **External Similarity** it means the semantic similarity among names and existing words in the dictionary (i.e., or domain words). To compute this metric, we proceed by calculating how similar names in a file are to their top three most similar words in the corpora. Finally, the external similarity is the median of the resulting similarities. For example, both classes in Figure 1 (`ExtractResources` and `MavenConsoleAnnotator`) has high external similarity because all names (e.g., `resource`, `charset`) indeed exists in the dictionary.

4. Empirical Study

We conducted an empirical study to quantitatively characterize the semantic similarity among names in large-scale open source-projects. In particular, we investigated how often developers use names that are semantically similar or names outside of the dictionary; and also whether the change history poses any effect on names semantic similarity across source code evolution and maintenance.

4.1. Goal and Research Questions

Our goal is to shed some light on the semantic similarity among names found in four large scale open-source projects. We used the organization proposed by the Goal/Question/-Metric (GQM) [Basili and Rombach 1988] to define the goals of our study. According

Table 1. Analyzed projects

Project	LoC	Num. Classes	Num. Attributes	Num. Variables
Tomcat	463178	3252	13629	24563
Spring-boot	376275	4576	11783	20754
Fastjson	199655	5139	16240	16693
Jenkins	320479	2016	6775	13845

to the proposed goal definition template, the scope of our study can be summarized as described below. In addition, based on our goal, we came up with three research questions (RQs) – presented below.

Analyze semantic similarity among names
with the purpose of characterization
with respect to attributes and variable names in object-oriented source code
from the point of view of the developers
in the context of five large-scale open source projects.

- **RQ$_1$:** Do developers use names that are semantically similar?
- **RQ$_2$:** Do developers use names outside of the dictionary?
- **RQ$_3$:** Do code change history exert influence on names semantic similarity?

We frame our discussion around our RQs and present the quantitative results in terms of the metric's internal similarity and external similarity (see Section 3.2). The studied subject projects (see Table 1) ranges from 199K to 463K lines of code. We have analyzed the amount of 14,983 classes and their contextual similarity considering 48,427 attributes and 75,855 variables.

4.2. Results and Discussions

4.2.1. RQ$_1$: Do developers use names that are semantically similar?

We resort to the internal similarity metric to characterize whether developers use names that are semantically similar. As mentioned in Section 3.2, the internal similarity metric varies from -1 to 1 (positive score is considered to have at least a reasonably good internal similarity). For all projects (see Table 2), we observed that developers use semantically similar names: the internal similarity median is around 0.30 ± 0.27. Moreover, more than 90% of all files have an internal similarity greater than 0 (zero). However, only a small

Table 2. Similarity scores table

Project	Min	Max	Median	Mean	Sd	-1	<-0.5	<0	>= 0	>0.5
Tomcat	-1	0.69	0.30	0.22	0.28	78	85	150	1754	4
Spring-boot	-1	1	0.34	0.32	0.15	21	22	65	2645	34
Jenkins	-1	0.71	0.30	0.24	0.22	18	23	100	936	9
Fastjson	-1	0.79	0.28	0.076	0.45	356	374	600	2103	11

Table 3. Most similar words score table

Project	Min	Max	Median	Mean	Sd	-1	<-0.5	<0	>= 0	>0.5
Tomcat	-1	0.96	0.76	0.72	0.22	27	27	41	1863	1863
Spring-boot	-1	0.90	0.77	0.76	0.12	9	9	24	2686	2686
Jenkins	-1	0.88	0.77	0.78	0.10	2	2	7	1029	1029
Fastjson	-1	0.92	0.76	0.72	0.21	35	35	51	2652	2652

fraction can be considered as having proper internal similarity (scores greater than 0.5). These results indicate that developers tend to always give good names to identifiers – ones that are at least a bit related between themselves. Only a small portion of the files have a negative internal similarity. In these cases, we consider that developers are not using existing words; or are using abbreviations and completely unrelated words. It is worth noting that an even smaller fraction of files turns out to have internal similarity equals to -1. In such cases, names are were all unrelated or even do not exists.

4.2.2. RQ$_2$: Do developers use names outside of the dictionary?

According to the word embeddings principle, existing words in the corpus will be always correlated to other words that also exist in the corpus, therefore, whether we managed to find words (in the corpus) similar to existing names in the source code, these names definitely exist in the dictionary. Therefore, we adopt the external similarity metric to understand whether developers use names out of the dictionary or not.

In Table 3 we can observe that the large majority of files have a positive score greater than 0.5. Therefore, we can conclude that developers in most cases resort to words that exist in the dictionary (median is around 0.77 ± 0.16). These results corroborate with Lawrie et al. [Lawrie et al. 2006] results showing that the use of dictionary words makes identifiers easier to read and understand. Indeed, only a small fraction of the files did not have positive scores. Files with external similarity equal to -1 have names that we could not find similar words in our subject corpus.

4.2.3. RQ$_3$: Do code change history exert influence on names semantic similarity?

During software development, developers make changes in the source code, either to add new features, to resolve existing bugs, or to improve code quality (e.g., refactoring). Inevitably, some of these changes may induce changes in the semantic similarity among

Table 4. Changes made up by authors

Projects	Changes		Authors	
	W	p-values	W	p-values
Tomcat	88536	0.001601	92080	0.02346
Spring-Boot	7891.5	0.002768	7944.5	0.003022
Jenkins	40424	7.07e-06	42933	0.0005925
Fastjson	580628	0.02746	542314	2,60E-05

names. Therefore, in order to better understand whether change history exerts influence on names semantic similarity, we use the `git` Log[3] capability to collect: (i) the number of changes per file; and (ii) the number contributor (i.e., authors) per file. To answer this question, we use the *Wilcoxon Rank Sum Test*. This test allows us to decide whether two populations (in our study, low and high semantic similarity) are *identical* or not without assuming that the populations follow a normal distribution. To ensure statistical significance, we adopted the customary $.05$ significance level ($p - value < 0.05$) for this test.

Table 4 presents the results that support this research question. We observe that changes in source code exert a statistically significant influence on names semantic similarity. Thus, the number of changes in a file is a promising indicator to distinguish between files with distinct levels of internal similarity – there is a tendency that the more developers modify a file in a project, the more they improve identifiers similarity. We also observed a statistically significant influence on names semantic similarity when we consider the number of distinct contributors. Therefore, assuming distinct developers contributing to the same files in a project can be a good practice to improve names and consequently the understandability of the source code.

5. Conclusion and Future Works

In this work, we resort to expand the current knowledge about identifiers by taking into consideration a novel dimension: the contextual similarity among them. To achieve such a goal we extracted and analyzed the semantic similarity among 139,265 identifier names distributed over four open-source projects. As a result, we were able to observe some patterns involving contextual similarity and the proper use of identifier names in source code. First, we could figure out that overall, classes tend to have at least a minimum level of contextual similarity; meaning that developers often name identifiers (e.g., variables/attributes) using words well aligned to the surrounding context. Second, the vast majority of the names used as identifiers can be found in the dictionary. This result might indicate the developer's awareness about naming conventions, even if they're not related to the rest of the class. Third, we could observe that files that suffer more changes tend to have better contextual similarity than those ones which do not change often. Considering the number of contributors per file, we observe the same pattern: those maintained by several contributors tended to preserve a better semantic similarity among identifier names.

As future works, we aim to continue studying the semantics similarity among identifiers and its effect on source code quality. Our objective is to observe the actual impact of contextual similarity in software development in maintainability. To do so, we intend to investigate how bugs are directly related to the absence of appropriated contextual similarity in classes. Moreover, we also set out to study to what extent contextual similarity exerts some influence on source code comprehensibility.

References

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

[3] git-scm.com/docs/git-log

Basili, V. R. and Rombach, H. D. (1988). The tame project: Towards improvement-oriented software environments. *IEEE Transactions on software engineering*, 14(6):758–773.

Butler, S., Wermelinger, M., Yu, Y., and Sharp, H. (2009). Relating identifier naming flaws and code quality: An empirical study. In *16th Working Conference on Reverse Engineering*, pages 31–35.

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

Deissenboeck, F. and Pizka, M. (2006). Concise and consistent naming. *Software Quality Journal*, 14(3):261–282.

Feitelson, D., Mizrahi, A., Noy, N., Ben Shabat, A., Eliyahu, O., and Sheffer, R. (2020). How developers choose names. *IEEE Transactions on Software Engineering*, pages 1–1.

Hofmeister, J., Siegmund, J., and Holt, D. V. (2017). Shorter identifier names take longer to comprehend. In *2017 IEEE 24th International conference on software analysis, evolution and reengineering (SANER)*, pages 217–227. IEEE.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Kawamoto, K. and Mizuno, O. (2012). Predicting fault-prone modules using the length of identifiers. In *2012 Fourth International Workshop on Empirical Software Engineering in Practice*, pages 30–34. IEEE.

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

Li, G., Liu, H., Liu, Q., and Wu, Y. (2018). Lexical similarity between argument and parameter names: An empirical study. *IEEE Access*, 6:58461–58481.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2017). Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.