A Large-Scale Investigation of Local Variable Names in Java Programs: Is Longer Name Better For Broader Scope Variable?

Hirohisa Aman[1][0000−0001−7074−5225], Sousuke Amasaki[2][0000−0001−8763−3457], Tomoyuki Yokogawa[2][0000−0001−6681−2608], and Minoru Kawahara[1][0000−0002−3542−5039]

[1] Ehime University, Matsuyama, Ehime 790–8577, Japan
{aman,kawahara}@ehime-u.ac.jp
[2] Okayama Prefectural University, Soja, Okayama 719–1197, Japan
{amasaki,t-yokoga}@cse.oka-pu.ac.jp

Abstract. Variables are fundamental elements of software, and their names hold vital clues to comprehending the source code. It is ideal that a variable's name should be informative that anyone quickly understands its role. When a variable's scope gets broader, the demand for such an informative name becomes higher. Although the standard naming conventions provide valuable guidelines for naming variables, there is a lack of concrete and quantitative criteria regarding a better name. That challenge in naming variables is the motivation of the quantitative investigation conducted in this paper. The investigation collects 637,077 local variables from 1,000 open-source Java projects to get a detailed view of the variable naming trend. The data analysis reveals frequently-used terms for variable names, the naming styles, and the length of names when the variable scopes are broad. The results showed that developers prefer to use fully spelled English words or compounded names for broad-scope variables, but they tend to avoid long names; Developers often use simple words or abbreviations shorter than seven or eight characters.

Keywords: Variable name · variable scope · Quantitative investigation.

1 Introduction

Variable names play essential roles in program comprehension [8, 11]. Meaningful and easy-to-understand names enhance the readability of the source code [10]. By naming variables appropriately, both the author programmers themselves and other developers can easily understand and review the source code [16]. On the other hand, making a variable's name hard-to-understand is a fundamental way of code obfuscation. We can easily obfuscate a source code by replacing all variable names with single meaningless characters [9]. Hence, a variable name can be a double-edged sword in terms of code quality because of its power of influence on the code readers.

The proper naming of variables has been attracting attention in the programming world. Many coding standards and practices mention the variable naming issue [2, 13, 16]. The standard rules and practices say that it is better to give a meaningful name to a variable. That is, it looks good to name a variable using a well-chosen English word or phrase (compounded words) describing the variable's role. On the other hand, developers sometimes prefer shorter names than fully-spelled English-word names or compounded ones. For example, many developers would favor "`i`" as the name of a loop counter rather than "`loop_counter`" if there is no chance of being misunderstood [2, 16]. Indeed, we often see many variables whose names are "`i`" in various source programs for such a use case.

Here, we focus on the opposite case, such that a short variable name like "`i`" is undesirable. A representative example is a case that the variable's scope is broad. We are questionable whether "`i`" is suitable for the name of a loop counter even when its scope is broader than several dozen or more lines of code. In such a case, programmers may prefer a longer and more descriptive name for the variable to make it more informative. However, to the best of our knowledge, there are no clear and concrete criteria to decide whether a variable name is proper or not when the scope is broad. If such criteria become available, we can build them into a static analysis tool or an advanced editor to evaluate the variable names and recommend better names in an automated way. That is a challenge in naming variables, and it motivated us to conduct a large-scale quantitative investigation of variables. In this paper, we report our investigation and discuss the results.

2 Related Work

Caprile and Tonella [8] prepared a standard dictionary of terms used for identifiers and a pre-defined naming grammar (a set of naming patterns), then proposed a method for detecting meaningless names and renaming them. Allamanis et al. [3] proposed a more enhanced framework that infers the coding convention adopted in a development project by mining the code repository. Their framework works to detect the undesired variable names (identifiers) violating the project's coding convention and to suggest better names for renaming such undesired ones by utilizing natural language processing techniques. Although the studies in [3, 8] provided helpful ways of improving variable names, the variables' scopes might also significantly impact their naming. Therefore, in this paper, we will conduct our investigation while considering the variable scope as well.

There have been large-scale investigations of variable names in the past. Beniamini et al. [6] focused on single-letter names such as "`i`" and investigated how programmers use those variables in their programs. They collected variable names from 1000 open-source projects whose languages include C, Java, JavaScript, PHP, and Perl. Although the single-letter name is the shortest name for a variable, their investigation revealed that some names could be meaningful in particular contexts, e.g., "`i`," and "`j`" as loop counters. Swidan et al. [17] conducted another investigation on Scratch programs and reported Scratch-specific variable naming trends: Scratch programmers tend to avoid single-letter names

for variables and prefer longer names with 4 – 10 characters. The investigation reported in [6] motivated us to analyze the variable naming trends further. Moreover, the data reported in [17] shows the possibility that the naming trends vary from language to language. Thus, we focus only on one programming language (Java) and conduct a further investigation of variable names while considering other aspects of variables, including the variable types and scopes.

There have been studies focusing on the length of the variable name. Although a longer name is easier to describe its role, developers get harder to memorize the name as it becomes longer in their programming and code review. Binkley et al. [7] conducted an experiment involving 158 programmers to examine the relationship between the variable name's length and the programmer's short-term memory and proved that too long names adversely affect program comprehension. Moreover, they pointed out that a longer name may raise the risk of choosing a wrong name when programmers use a program editor's auto-completion function. Hofmeister et al. [15] and Aman et al. [4, 5] reported empirical results showing that long names may have harmful impacts on program comprehension and fault-proneness. Thus, we have considered that a long and descriptive variable name is not always the best, even if its scope is broad. That thought is also one of our motivations for conducting a large-scale investigation.

3 Quantitative Investigation

We conducted a large-scale quantitative investigation regarding variable names in Java programs. In this section, we report and discuss the results.

3.1 Aim and Surveyed Software

This study aims to capture the features of broad-scope variable names through a data collection. Moreover, we can get a corpus of terms used in variable names by organizing the collected data. Such a corpus would help an automatic evaluation of variable names and a recommendation toward better namings. We discuss the following two research questions by analyzing the collected data.

RQ1: What are the features of the variable names when their scopes are broad?
RQ2: What kind of terms do developers use for broad-scope variable names?

As we mentioned in Section 2, we collect variable data from only Java software to avoid the impact caused by the difference in the programming language. Moreover, we focus only on the local variables[3] in this study because the names of the global variables, i.e., *fields* in Java, tend to depend on the class design, and the programmers may have no power to decide their names. Since local variables are available only within a method, the naming would usually be the programmer's discretion. We believe a large-scale data collection of variable names would be worthwhile to understand the naming trend and the programmers' preference regarding broad-scope variables' names. In this study, we survey (collect data from) 1,000 Java open-source projects having high "stars" ranks at GitHub.

[3] The local variables include the formal arguments (parameters) of methods.

3.2 Results of Data Collection

As a result, we successfully got 472,665 Java source files from 971 projects. We could not analyze source files from 29 out of 1,000 projects due to one of the following three reasons: (a) the project had no Java source file (5 projects), (b) all source files were the ones to be excluded (e.g., test programs) (22 projects), and (c) our source file analysis failed because the source file path included a multi-byte character (2 projects). We could extract 637,077 local variables, i.e., the local variable declarations or formal argument declarations in methods.

Table 1 presents the distribution of variables' scope length. The scopes of the most variables were shorter than about 40 lines; the 90 percentile was 42 lines. The method length would cause this result because the local variables are available only within the method. On the other hand, we found variables whose scope lengths are broader than 3,000 lines. We manually checked the details of the variables with the broadest top 10 scopes. Then, we revealed that such ones appeared in a simple but huge method which only performs many assignments to an array. Such methods seem to be automatically generated code. Thus, we decided to exclude local variables with too-broad scopes from our analysis as outliers. Consequently, we use the variables whose scope ranges are between 90 and 99 percentiles (42 and 157 lines) as our sample set of broad-scope variables.

3.3 Results of Variable Name Categorization

Next, we report the results of our heuristic categorization of variable names. We began with four categories of variable names: i) single letter names, ii) dictionary-word ones, iii) compounded ones, and iv) other names. Then, we found 62,236 other names (9.8%), and we manually checked their names to explore a better categorization. Then, we detected the following six exceptions.

(a) Technical terms: We observed 131 technical terms that we usually see in programs, but Aspell's dictionary does not include them: for example, "default," "git," and "setter." We prepared a user dictionary that complements the default dictionary for capturing the above technical terms.

(b) Type-derived names: We encountered the variable name "`fos`." That name seems to be derived from its type, i.e., the class name, "`FileOutputStream`." Through our additional exploration, we also found the variable names that contain the type name as a substring (e.g., "`tLabel`" whose type is "`Label`"). Moreover, we noticed that there might be the plural form of a type-derived name such as "`mqs`" whose type is "`Set<MessageQueue>`"; It may mean that the variable contains two or more "`MessageQueue`" objects. Hence, we also check the above string matching for the name by dropping the trailing "`s`."

Table 1. Summary statistics of scope length.

	percentile (unit: source lines)									
min	10	20	30	40	50	60	70	80	90	max
1	2	3	4	6	8	11	16	24	42	3,680

(c) Abbreviated words: Before our data collection, we expected to encounter local variables whose names are abbreviated words. However, we did not initially provide the name category of abbreviated words because it is challenging to prepare the complete list of abbreviations. Instead, we continued updating our abbreviation dictionary by checking the names included in the "other names" category to build the additional name category, "abbreviated words." When we update the dictionary, we checked if the abbreviated word is common or not by referencing https://www.abbreviations.com/. Our abbreviation dictionary consists of 201 terms, and it is available from our data website. We accept the plural form of abbreviations (e.g., `args`) as the ones being in this category.

(d) Numbered names: We saw many "numbered" names like "`x2`." Developers would add numbers to the end of variable names to declare two or more variables whose roles are similar. Any name in any category can become the base name of a numbered variable name, including single letter names, dictionary word ones, abbreviated word ones, compounded ones, and type-derived ones.

(e) Variants produced by adding an extra character: We also saw variants of dictionary names, which are produced by adding an extra character to the head or the end of the words (e.g., *f*`name`, `cell`*x*). We consider an additional category for such variants and denote it by "dict word name +1." We also introduce similar "+1" categories to the remaining categories to cover any variants.

(f) Concatenated names: There were exceptional names made by merely concatenating two words, such as "`filename`." Because there is no character-case change nor delimiter in such concatenated names, we implemented the following simple algorithm to fix the name category: if a name can be split into two substrings and both are in the English dictionary, our technical term dictionary, or our abbreviation dictionary, then we regard the original name as a compounded name. Although the above algorithm cannot work for a potential compounded name made by concatenating three or more words, there are almost no such names in our dataset, so we did not adopt a more sophisticated algorithm [12].

We have updated our variable name categories through heuristics, considering the above exceptions. Consequently, we made 17 categories of variable names shown in Table 2. Notice that we did not prepare the "+1" category (variant name category) for the single letter names (No.1–2) and the compounded names (No.15–16). In the case of a single letter name, it is hard to decide which character is the base name. For compounded names, we can split them into sub names. Then, the split sub names can also be classified into the remaining name categories. Although we could have divided the compounded name category by applying the remaining ones recursively, we avoided making the category organization too complex.

As a result, we classified the collected 637,077 local variables into the above 17 name categories, as shown in Table 2 (see "all variables" column). The most major categories are No.3:dictionary word names (39.5%), No.15:compounded names (34.2%), and No.1:single letter names (17.2%). In the table, we also show the frequencies of names when we focus only on the broad-scope variables whose scope lengths are between 42 and 157 lines (see "broad scope only" column). For

Table 2. Local variable name category and frequency of names by category.

No.	category	frequency (%) all variables	frequency (%) broad scope only	example
1	single letter	109,471 (17.2%)	2,884 (5.1%)	`i`
2	single letter + num	7,092 (1.1%)	1,074 (1.9%)	`t1`
3	dict word	251,723 (39.5%)	17,982 (32.1%)	`result`
4	dict word + 1	3,328 (0.5%)	464 (0.8%)	`stepx` (step + x)
5	dict word + num	5,422 (0.8%)	742 (1.3%)	`count5`
6	dict word + num + 1	109 (0.0%)	28 (0.0%)	`inputp1` (input+p+1)
7	type derived	19,942 (3.1%)	1,578 (2.8%)	`fos` (FileOutputStream)
8	type derived + 1	1,097 (0.2%)	66 (0.1%)	`jconf` (j+Configuratin)
9	type derived + num	433 (0.1%)	37 (0.1%)	`str2`
10	type derived + num + 1	26 (0.0%)	1 (0.0%)	`tvecs2` (t+Vec[ ] +2)
11	abbrev word	10,625 (1.7%)	778 (1.4%)	`buf`
12	abbrev word + 1	433 (0.1%)	37 (0.1%)	`imgw` (img + w)
13	abbrev word + num	486 (0.1%)	29 (0.1%)	`tmp2`
14	abbrev word + num + 1	16 (0.0%)	0 (0.0%)	`ctrly1` (ctrl+y+1)
15	compounded	218,138 (34.2%)	28,758 (51.3%)	`commaIndex`
16	compounded + num	2,910 (0.5%)	875 (1.6%)	`upperString2`
17	other	5,826 (0.9%)	727 (1.3%)	`xy`
	total	637,077	56,060	

the broad-scope 56,060 variables, the most major category was the compound names, and about half of the variables belong to this category. The second and the third most major categories were the dictionary word names (32%) and the single-letter names (5%), respectively. Although we also observed type-derived names, abbreviated-word names, numbered names, and variants ("+1" names), all of their rates are about or less than 3%, so they seem to be the minorities in the Java local variable names regardless of their scope lengths.

3.4 Discussion

We discuss the results from the perspective of our RQs.

RQ1 (What are the features of the variable names when their scopes are broad?) We regarded the variables whose scopes are between the 90 and 99 percentiles as the broad-scope variable samples. About half of the variables' names are compounded names, and about one-third of the ones are single dictionary (fully spelled) words, as we saw in Table 2. Developers seem to prefer such names when the variables' scopes get wider. Fig. 1 presents the comparison of the name category shares between "all variables" and "broad-scope variables." Table 2 and Fig. 1 show that the following three name categories' shares are relatively specific to the broad-scope variables' names: No.1 (single letter), No.3 (dictionary word), and No.15 (compounded names). While the shares of the single letter names and the dictionary ones decrease, the share of compounded

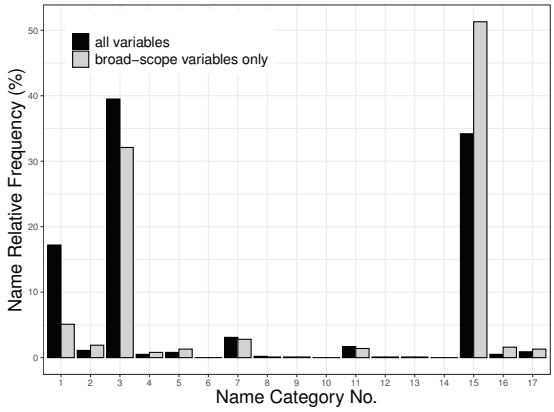

Fig. 1. A comparison of name category shares: all variables vs. broad-scope variables.

names increases when the scope is broad. The remaining 14 categories' shares did not show large differences between "all variables" and "broad-scope variables." Thus, the broader scope would lead developers to adopt more descriptive compounded names rather than single-letter names and single-word names. Although developers also use single-letter names, its share is only about 5%. We additionally checked broad-scope variable samples with single-letter names and revealed that the symbolic names are "`i`" and "`v`." A variable "`i`" tends to play a control variable of a loop or an index of an array, and a variable "`v`" seems to be used for meaning a "value" of an object or a "vector" of objects.

Because developers tend to use compounded names for broad-scope variables, their variable name lengths may become longer than the others. We additionally examined the distribution of variable name length by counting the number of characters comprising the variable names. Fig. 2 shows a comparison of the variable name lengths. In the figure, we considered three sets of variable names:

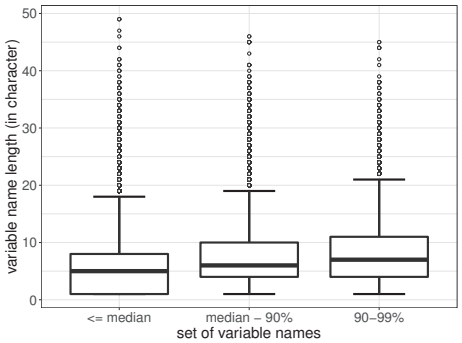

Fig. 2. A comparison of variable names' length.

(1) The variables whose scope lengths are equal to or narrower than the median; (2) The variables whose scope lengths are between the median and the 90 percentile; (3) Our broad-scope sample set, i.e., the variables whose scope lengths are between the 90 and 99 percentiles.

Although there is a gradually increasing trend as the variable scope gets broad, their differences are minor; the medians of variable name lengths are 5, 6, and 7, respectively. We cannot say the broad-scope variable has a notable long name. When a variable's scope gets broader, developers are likely to make the variable name more informative by successfully combining simple words or abbreviated words while controlling variable name length.

Therefore, we answer RQ1 as follows: Developers seem to name the broad-scope variables using compounded terms or fully spelled words, but they tend to avoid long names. The median name length is seven characters. Although single-letter names are not well-used, "`i`" and "`v`" are exceptions.

RQ2 (What kind of terms do developers use for broad-scope variable names?) To comprehend the trend of words used for broad-scope variables' names, we tallied up the terms appearing in those variable names. Table 3 presents the most frequently-appearing terms (top 50), where the portions of a compounded name are also included; For example, when we have a compounded name "`toString`," we split it into "`to`" and "`string`" and tally up each of them.

The top 50 terms consist of 42 dictionary words, six abbreviations (`id`, `max`, `num`, `pos`, `info`, `url`), and two single letters (`i` and `v`). Here, we see that the length of the most terms (46 out of 50) is shorter than seven characters. Because well-used abbreviations "`id`" and "`info`" may be from "identifier" or "identification" (10 or 14 characters) and "information" (11 characters), developers might use shorter names instead. Interestingly, both "`pos`" and "`position`" ranked in the top 50 list. Some developers might consider the word "position" (8 characters) to be a bit longer for a variable name and preferred "`pos`." That is, such a length may be a threshold to decide whether the name looks long or not, and this result corresponds to the trend of broad-scope variable's name length (the median is 7; see Fig. 2). Notice that our main focus of the above discussion is on the name length and not on the semantics. Because different programmers might use the same term to represent different concepts, the results might mix up such terms.

We further checked the details of 28,758 compounded names used for broad-scope variables. Then, we found that about 87% of compounded names consist

Table 3. Most frequent terms used in single names or portions of compounded names of broad-scope variables (Top 50).

`name`	`type`	`id`	`index`	`count`	`start`	`i`	`size`	`width`	`list`
`file`	`result`	`data`	`view`	`height`	`is`	`v`	`max`	`time`	`end`
`offset`	`value`	`path`	`to`	`current`	`num`	`key`	`field`	`line`	`new`
`length`	`text`	`item`	`last`	`in`	`pos`	`map`	`info`	`child`	`position`
`class`	`action`	`first`	`builder`	`left`	`request`	`url`	`color`	`string`	`layout`

Table 4. Top 10 abbreviations.

num	idx	params	buf	src	args	tmp	param	arg	ret

of only dictionary words, and about 6% of them are mixups of dictionary words and abbreviations. That is, developers prefer to use dictionary words even in a compounded name. Table 4 presents the top 10 abbreviations, which appeared in broad-scope variable names. The abbreviations "`param`" and "`params`" would be valuable examples showing the above trend of avoiding longer names. Although we can consider a similar reason for "`args`," they may also have another reason: "`args`" often appears as the default parameter of "main" method in Java.

Therefore, we answer RQ2 as follows: For broad-scope variables, developers prefer fully spelled words to make their names informative even in a compounded name. However, they also tend to avoid making the name longer. The threshold of character counts would be seven or eight. When the name gets longer, developers will likely to replace the fully spelled words with their abbreviations. They often use short and meaningful terms shown in Tables 3 and 4.

Although our investigation is preliminary work on variable names, we could prove the trend that many programmers avoid long names even if their variables' scopes are broad. Such a trend seems to correspond to the harmfulness of too long names pointed out by Binkley et al.[7]. Moreover, we successfully found a quantitative baseline of name length to judge if a name is long or not. We plan to design an automated system for evaluating and recommending variable names using our findings and our dataset[4] of variable names in the future.

3.5 Threats to Validity

We describe threats to validity in our study below.

Construct Validity: We measured the length of a local variable's scope by focusing on the starting line number and the ending line number of the variable's scope in the source file. For example, when a variable is declared at the 15th line and is valid until the 26th line in the source file, we regard the length of the variable's scope as 12 lines (=26-15+1). Because it might also count the comment lines and blank lines, the truth scope lengths might be slightly shorter than the values we reported in this paper. In addition to the camelCase names (e.g., `fileName`) and snake_case names (e.g., `file_name`), we classified the two-term concatenation names (e.g., `filename`) into the compounded name category as well. However, our classification cannot treat the exceptions made by concatenating three or more terms. We further checked the "other names" category and found 25 exceptional variable names like "`linestartpos`" which may be "`line`" + "`start`" + "`pos`"(position). We might successfully classify such exceptions if we utilized more sophisticated variable (identifier) splitting studies [14]. However, those exceptional names are less than 0.1% of the studied names, so we consider they have almost no impact on our findings.

[4] Our dataset is available from https://bit.ly/3xLuaLK.

Internal Validity: To automatically collect local variable data in this study, we developed a program analysis tool using the Eclipse Java development tools (JDT) [1]. Due to the version of the JDT parser we used, our tool does not support the Java lambda expressions using the arrow operator "`->`." That is, the tool misses the data on local variables appearing in the functional programming context. Although our dataset includes most of the local variables used in Java programs, such a lack of data is a threat to our study's internal validity.
External Validity: We conducted a large-scale investigation of local variable names by collecting the data from many open-source software projects. Because we chose popular projects without any bias in the project domain and scale (numbers of source files and developers), we believe our dataset is a sample set of the general trend regarding local variable names. However, our data collection is limited to Java open-source software. That is, our findings might not be generalized to commercial software or other programming language software.

4 Conclusion and Future Work

In this paper, we have focused on local variables' names in Java programs. Although programmers often use single-letter names for local variables, the naming trend might change to make a variable's name more informative when the variable's scope gets broader. To understand the naming trend for broad-scope variables, we conducted a large-scale quantitative investigation of local variable names with the following two research questions.

RQ1: What are the features of the variable names when their scopes are broad?
RQ2: What kind of terms do developers use for broad-scope variable names?

For RQ1, we found that developers tend to use compounded names or fully-spelled ones for broad-scope variables. Meanwhile, they are likely to avoid making long names; The median name length is seven characters. Although single-letter names are not well-used, "`i`" and "`v`" are exceptions. For RQ2, we affirmed that developers prefer short and meaningful dictionary words to make their names informative even in a compounded name. Nonetheless, they also tend to avoid long names, and the threshold of character counts would be seven or eight. When a variable's name gets longer, developers seem to shorten it by replacing the fully-spelled words with their abbreviated words or shorter other terms.

Although longer variable names would be more informative or descriptive, our investigation results proved that longer names are not always the better names for broad-scope variables. Choosing simple but meaningful terms would become the core of beneficial variable naming. Our data would be useful in an automated evaluation of variable names and a recommendation of proper names for enhancing the code readability; Our dataset is available from https://bit.ly/3xLuaLK.

Our future work includes applications of the investigation results into automatic aid for enhancing code readability. The naming trend and the variable name corpus would help enrich static code analysis tools (code analyzers). We plan to develop a code-review supporting tool that detects problematic variable names and suggests better alternatives based on our survey dataset.

Acknowledgment

This work was supported by JSPS KAKENHI #20H04184, #21K11831, and #21K11833.

References

1. Eclipse Java development tools (JDT). https://www.eclipse.org/jdt/
2. Linux kernel coding style. https://www.kernel.org/doc/html/v4.10/process/coding-style.html (2016)
3. Allamanis, M., Barr, E.T., Bird, C., Sutton, C.: Learning natural coding conventions. In: Proc. 22nd ACM SIGSOFT Int. Symp. Foundations of Softw. Eng. pp. 281–293 (Nov 2014)
4. Aman, H., Amasaki, S., Sasaki, T., Kawahara, M.: Empirical analysis of change-proneness in methods having local variables with long names and comments. In: Proc. 9th Int. Symp. Empir. Softw. Eng. & Measurement. pp. 50–53 (Oct 2015)
5. Aman, H., Amasaki, S., Yokogawa, T., Kawahara, M.: Local variables with compound names and comments as signs of fault-prone java methods. In: Joint Proc. 4th Int. Workshop on Quantitative Approaches to Softw. Quality & 1st Int. Workshop on Tech. Debt Analytics. pp. 4–11 (Dec 2016)
6. Beniamini, G., Gingichashvili, S., Orbach, A.K., Feitelson, D.G.: Meaningful identifier names: The case of single-letter variables. In: Proc. 25th Int. Conf. Program Comprehension. pp. 45–54 (May 2017)
7. Binkley, D., Lawrie, D., Maex, S., Morrell, C.: Identifier length and limited programmer memory. Sc. Comp. Programming **74**(7), 430 – 445 (May 2009)
8. Caprile, B., Tonella, P.: Restructuring program identifier names. In: Proc. Int. Conf. Softw. Maintenance. pp. 97–107 (Oct 2000)
9. Ceccato, M., Di Penta, M., Falcarin, P., Ricca, F., Torchiano, M., Tonella, P.: A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. Empir. Softw. Eng. **19**(4), 1040–1074 (Aug 2014)
10. Corazza, A., Martino, S.D., Maggio, V.: Linsen: An efficient approach to split identifiers and expand abbreviations. In: Proc. 28th Int. Conf. Softw. Maintenance. pp. 233–242 (Sept 2012)
11. Deissenboeck, F., Pizka, M.: Concise and consistent naming. Softw. Quality J. **14**(3), 261–282 (Sept 2006)
12. Enslen, E., Hill, E., Pollock, L., Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis. In: Proc. 6th Int. Working Conf. Mining Softw. Repositories. pp. 71–80 (May 2009)
13. Gosling, J., Joy, B., Steele Jr., G.L., Bracha, G., Buckley, A.: The Java Language Specification. Addison-Wesley, Boston, MA (2014)
14. Hill, E., Binkley, D., Lawrie, D., Pollock, L., Vijay-Shanker, K.: An empirical study of identifier splitting techniques. Empir. Softw. Eng. **19**(6), 1754–1780 (Dec 2014)
15. Hofmeister, J., Siegmund, J., Holt, D.V.: Shorter identifier names take longer to comprehend. In: Proc. 24th Int. Conf. Softw. Analysis, Evolution & Reeng. pp. 217–227 (Feb 2017)
16. Kernighan, B.W., Pike, R.: The practice of programming. Addison-Wesley Longman, Boston, MA (1999)
17. Swidan, A., Serebrenik, A., Hermans, F.: How do scratch programmers name variables and procedures? In: Proc. 17th Int. Working Conf. Source Code Analysis & Manipulation. pp. 51–60 (Sept 2017)