2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC) | 979-8-3503-3750-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPC58990.2023.00037

Reanalysis of Empirical Data on Java Local Variables with Narrow and Broad Scope

Dror G. Feitelson
Dept. Computer Science
The Hebrew University, Jerusalem, Israel

***Abstract*—It is generally accepted that variables with a narrow syntactic scope can have short names, whereas variables with a broad scope require more informative longer names. We study how names are given in practice, using a dataset of nearly 640 thousand variable names from Java methods, recently introduced by Aman et al. We extend their original analysis by using a finer division of scopes into ranges. We find that indeed variables with broader scope tend to be slightly longer and to include more words. There is also a progression of changes in name structures, with fewer single-letter names and more compound names as the scope increases. But the biggest differences occur at the low-scope end, not the high-scope end. In addition, we present more evidence that words of 6 letters or more are often abbreviated, but this is not affected by scope. Finally, we also analyze the distribution of popularity of names and of words in names, and show that single letter names are much more varied and common than usually thought, even when the variables have a broad scope.**

***Index Terms*—variable name, variable scope, name length**

I. Introduction

The names of variables and other code elements often provide the only clues about the purpose of a computer program. When we read a block of code in which a function is applied to the items in a collection, it is the names which tell us whether this code updates the grades of students in a course or adds a bonus to the salaries of employees in a company. In particular, informative names are a central element of the "clean code" approach, which advocates code that can be read like prose with no additional documentation [27].

Due to the importance of names, a basic research question is what exactly defines a "good" name. Considerable research has been devoted to this question. For example, Deissenboeck and Pizka formulized rules for when names are concise and consistent [12], and Arnaoudova et al. identify antipatterns that reflect inconsistencies between names and how they are used [4]. Fakhoury et al. later showed that using names with such antipatterns leads to higher cognitive load when reading the code [14]. Binkley and Lawrie and Caprile and Tonella consider the normalization of the vocabulary used to construct names [7], [9]. Avidan and Feitelson find examples of misleading names which interfere with code comprehension [5], while Beniamini et al. show that even single-letter names may convey meaning [6]. Feitelson et al. suggested a 3-step model of how developers choose names, and show that using this model leads to the creation of longer and better names [16].

A recurring issue in the research literature is the length of names. Several papers have compared the use of full words, abbreviations, and single letters in names in terms of their effect on code comprehension [24], [28], [19]. Binkley et al. note that name lengths should also be limited because longer names would be harder to remember [8]. Etgar et al. show that focused information assists in remembering names, but that the best recollection is achieved for short names [13]. Aman et al. claim that methods with longer names tend to be more bug-prone [1], but a study by Lemos et al. indicates that longer names reflect engineered code [25]. Data collected by Holzmann indicates that function and variable names are becoming longer with time [20].

The focus in our work is on the empirical relationship between local variable name lengths and their syntactic scope. It is commonly thought that short names can be appropriate when variables have a limited scope, because in this situation the meaning is implied by the immediate context. Moreover, misunderstanding a local variable's use is expected not to have consequences beyond the surrounding block of code. For example, Kernigham and Pike write that "shorter names suffice for local variables; within a function, `n` may be sufficient, `npoints` is fine, and `numberOfPoints` is overkill" [22]. They then continue by giving an example where a loop is much clearer when abbreviated names are used, and conclude "Programmers are often encouraged to use long variable names regardless of context. This is a mistake: clarity is often achieved through brevity." Using short names in a local scope is also advocated by the Free Software Foundation's *Gnu coding standards*[1] and by Oracle's *Code conventions for the java programming language*[2].

However, it seems that there has been very little research on this issue. Liblit et al. report that the lengths of names of local variables are considerably shorter than those of functions or global variables [26]. More recently Aman et al. performed a focused study on whether longer variable names are actually used in larger scopes [2]. This study made two important contributions. First, it created a large dataset with hundreds of thousands of variable names from nearly 1000 popular Java projects from GitHub, including an analysis of their

This research was supported by the ISRAEL SCIENCE FOUNDATION (grant no. 832/18).

[1] https://www.gnu.org/prep/standards/#Names
[2] https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

2643-7171/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPC58990.2023.00037

composition and information about their scope. Second, it laid the groundwork for analyzing the relationship between name structure and various other attributes (including scope), and identified compositions which have escaped scrutiny so far—and specifically, the inclusion of abbreviations, numbers, and single letters within compound names.

The results presented by Aman et al. indeed show a correlation between name length and scope. And this correlation depends on the names' structures: it is positive for compound names, but negative for single-letter names, and, to a lesser degree, single word names. But these results are based on a rather crude classification of scopes, singling out "broad scope" (defined as the 90–99th percentiles of the scope distribution) compared to all other scopes. We perform a more detailed analysis based on the same data, and conclude that name types exhibit a progression as a function of scope. We also see that the most divergent behavior occurs not in the broadest scope but in the narrowest scopes, which we define as consisting of up to 5 lines. In addition, we study the distribution of words which are abbreviated, and show that abbreviations tend to be applied to words of length 6 letters and above. This constitutes a slight adjustment to the observation from the original study which noted that programmers tend not to use words of 7 letters or more, but without showing an analysis. Finally, we present a more detailed analysis of the distribution of popularity of names and of words in names, and show that many different single-letter names are used, even when the variables have a broad scope.

II. The Aman et al. Dataset

Aman et al. conducted a study where they collected data about 637,077 local variables from 472,665 files of 971 open-source Java projects stored on GitHub [2]. Initially they set out to collect data from the 1000 top starred projects, but 29 of them did not provide usable data.

Aman et al. provide access to their dataset on the web at URL https://bit.ly/3xLuaLK. The main file is categorized_local_vars.txt. This is a large space-separated file with a line for each variable. The important fields for us are:

- **Field 3: name** — the variable name. This is the variable as it appears in the code, including capitalization used to distinguish words in camelCase style.
- **Field 7: range** — the scope of the variable in lines. This is calculated as the last line where the variable is valid minus the line where it is declared (so a scope of 1 means the declaration plus one line).
- **Field 9: words** — the partitioning of compound names into words. This is largely based on identifying camelCase notation, but they also applied a function to recognize the concatenation of two dictionary words. Unfortunately this sometimes led to errors. For example, we observed a case where "throwable" was partitioned into "throw" and "able".
- **Field 11: details** — a classification of the words in the name, including:
  - **ch** — a single character.
  - **dict** — a dictionary word, found in the dictionary of the aspell spelling utility. In addition 131 technical terms were added to the dictionary manually after not being recognized, for example "git" and "setter". This additional dictionary is provided with the data. Using dictionary words is often considered desirable; for example, Lawrie et al. even define identifier quality by the use of dictionary words or a small set of well-known abbreviations [23].
  - **abbrev** — an abbreviation. These were identified manually from the words that were not identified otherwise, and their list is also provided with the data.
  - **type** — a data type or derived from a data type (e.g. an acronym, like "cdl" for CountDownLatch).
  - **num** — a number which is embedded in the name.
  - **other** — none of the above, e.g. a sequence of letters that is not a word but was also not identified as an abbreviation, like "inv".

  Foe example, the made-up name q1code would be divided into 3 with the classifications ch, num, and dict.

  Dictionary words were by far the most common classification. However, the dictionary used by Aman et al. may be overly permissive, as we observed cases where a single letter (other than "a" or "i") or what we would consider an abbreviation were recognized as a dictionary word.

The data also contains a classification of names into 17 different categories, based on the combination of word types they include (for example, a dictionary word and a number and a single character). Aman et al. use these classifications in their analysis, but most of the categories have relatively few names (only 6 categories contain more than 1% of the names each, and the top three—dictionary word, compound, and single letter—account for 91% of all names). We do not use these combination categories.

III. Analysis and Results

Aman et al. divided the range of scopes into percentiles and deciles [2]. They excluded the top percentile, and report most of their results by comparing the remaining top decile (the 90-99th percentiles) with the full data. In the case of length distributions they compare with names below and above the median scope. These partitionings are problematic for two reasons. First, they are coarse. Second, they induce mixtures. For example, the 30th percentile is a scope of 4 lines. This means that variables with such a scope appear in both the 3rd and the 4th deciles.

In contrast, we define our partitioning based on roughly-logarithmically-spaced boundaries, as described in Table I. This leads to groups that are not equally sized, but they do fall in the range of 10–20% of the names each. A special issue is the top boundary. The data contains scopes up to 3680 lines, but Aman et al. report that the top ones appear to be from automatically generated code. They therefore arbitrarily decided to exclude the top percentile of the data, leading to a top accepted scope of 157 lines. We set the boundary higher,

TABLE I
Partitioning of scopes into groups.

Scope	*Names*	*Percent*	*Comment*
≤2	125,513	19.7%	minimal
3–5	124,487	19.5%	narrow
6–10	120,208	18.8%	
11–20	115,426	18.1%	
21–40	84,832	13.3%	
41–380	65,678	10.3%	broad
>380	933	0.1%	excluded

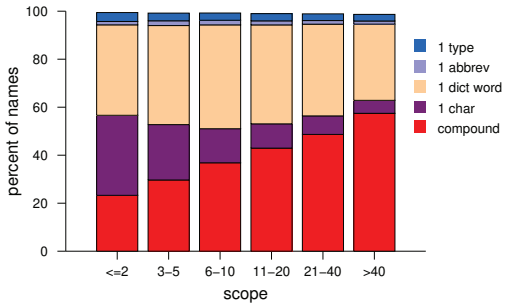

Fig. 1. *Classification of different name types in different scope groups.*

at 380 lines. The justification is that functions in the 200–300 lines range are known to be written manually. The reason for choosing 380 lines as the limit is that when viewing the distribution of scopes in the data, this is the first point of discontinuity. This boundary leads to the exclusion of only 0.1% of the data.

A. Name Types and Composition

Using the above partitioning, Figure 1 shows the name classes (as identified by Aman et al.) that appear in each range of scopes. Two trends stand out. The first is the rise in using compound names in larger scopes. In the first scope group, where the scope is up to only 2 lines, only 23.3% of the names contain multiple terms. In the last group, with scopes above 40, this rises to 57.4% of the names. So developers obviously prefer using compound names in larger scopes. This may be due to a perception that they are more informative and aid code comprehension [29].

Single-term names naturally make up the remainder of the names. They therefore go down from being 76.7% of the names that have minimal scope to 42.6% of the names that have broad scope. The main difference occurs with single-letter names, which go down from a third of all names (33.3%) to just 5.4%. Dictionary words form roughly 40% of the names in all the groups, a bit more for medium scopes (with a maximum of 43.3% for scopes of 6–10), and significantly less only in broad scopes (31.7%). Type-based names and standalone abbreviations are rather rare and their numbers hardly depend on scope.

These results conform with those of Aman et al., but provide greater detail. For example, we can see that the

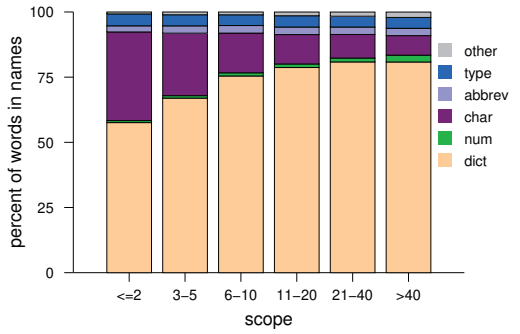

Fig. 2. *Word types appearing in the names in different scope groups.*

broad-scope names and slightly less-broad-scope names have similar distributions, and that the main difference occurs in narrow-scope names, where single-letter names are especially plentiful. Also, we see that their result concerning the reduced number of dictionary-word names in the broad-scope group is only part of a wider pattern.

The same type of analysis can be applied to the words in compound names. To do so we divide these names into their constituent terms, and count the different types. It turns out that most of the words in compound names are dictionary words (85.4–87.6%), and that there are only very minor differences between scope groups with no significant trends. We then combine this with the standalone terms (that is, non-compound names with one word). The result is shown in Figure 2. This is dominated by the two trends we saw already: increasing dictionary words and decreasing single letters with increased scope. Numbers and other also appear more in large scopes, but their absolute numbers are low.

We note that the low number of type-derived names (around 3%) is much lower than the result of Gersta et al., who found that 11% were related to types (their "ditto", "diminutive", and "cognome" categories relative to the complete 1.4 million names analyzed) [17]. The difference could be explained by the fact that they also consider multi-word type names. However, this explanation would require that types be common among the words making up names, which is not the case (they are a bit more than 4%). So we have no explanation of this difference in results.

B. Name Lengths

Turning to the length of variables, Figure 3 shows the average length of variable names as a function of scope, measured in characters and in words. The data shows a steady upwards trend, from 1.3 words and 5.0 letters at the minimal scope up to 1.8 words and 8.3 letters at a broad scope. This again provides more detail than the results presented by Aman et al.

Another way to look at the relation between name length and scope is to calculate the correlation coefficient. Given that there is no reason to think that the relation is necessarily linear,

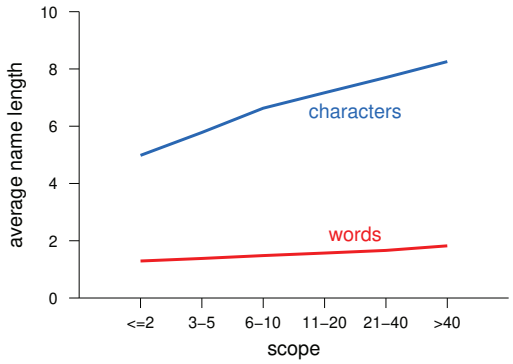

Fig. 3. *Average length of variable names as a function of scope.*

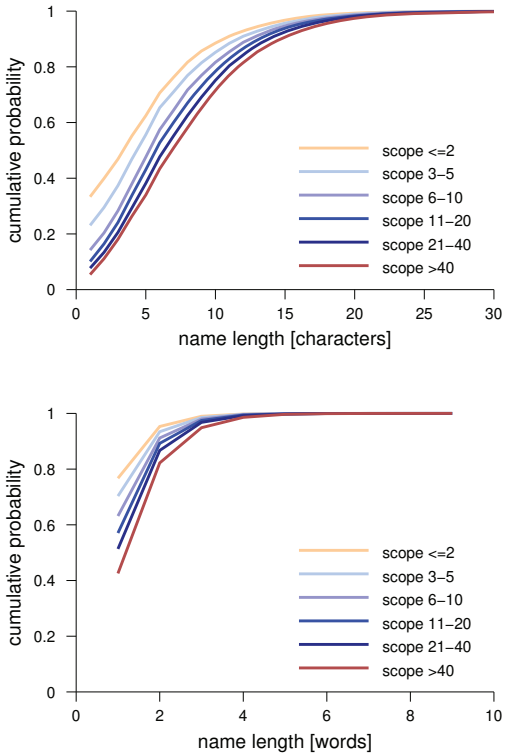

Fig. 4. *CDFs of the distributions of names' lengths in the different scope groups, where length is counted in characters and in words.*

we use the Spearman correlation coefficient, which quantifies the degree that the relationship is monotonic. The result is that the Spearman correlation of scope and length as measured in letters is 0.24, and when measured in words it is 0.22. These are positive correlations, but rather low, indicating a diffuse distribution of values.

Figure 4 shows the actual distributions. The plots are cumulative distribution functions (CDFs), showing for each length the fraction of names that are up to this length. A graph that is shifted to the right relative to another indicates that the distribution tends to higher values. The resulting distribution functions are cleanly separated from each other, implying that at each percentile the values grow larger with scope. The differences are larger for narrow scopes, which as we saw have many single-letter names.

Figure 5 looks at name length as measured by characters and words together. These are just histograms of the name lengths as measured in characters, but each bar is colored according to the number of words that compose the name.

It is easy to see again that single-letter names are especially common in low scopes. In addition, even the longer names that appear in these scopes tend to have only one word: in the minimal scope of $\leq$2 lines only 23.3% of the names have more than one word, and for scopes of 3–5 it is only 29.7% of the names. But for larger scopes the number continues to rise, reaching 57.4% of the names for the broadest scope as noted above. In addition, there are also more names with more than two words.

If we draw a single such histogram for all the names in all the scopes, the resulting distribution of lengths is bi-modal. The first mode is the single-character names common in the small scopes. The second mode, peaking at lengths of 4–6 letters, comes mainly from the middle and large scopes This mode is mostly composed of single-word names, but it has a wide right tail, which is dominated by two- and three-word names. This distribution is similar but not the same as the one found by Beniamini et al. 5 years ago, based on the top 200 Java projects in GitHub [6]. In their case the single-letter mode was less pronounced, and the second mode was wider, peaking at lengths of 4–8 letters.

C. Use of Abbreviations

One of the observations made by Aman et al. in their study was that 46 of the 50 most-used terms in names were shorter than 7 characters. In addition, they noted that some of these terms were abbreviations of words with 10 to 14 letters. They therefore speculated that programmers tend to abbreviate words that are 7 characters long or more.

To verify this we analyzed the list of all abbreviations identified in Aman et al.'s dataset. We classified these abbreviations into three classes. The first were abbreviations we could identify. For these we noted the original full words that had been abbreviated. Note, however, that the relationship of words to abbreviations is many-to-many. There are cases where the same word has multiple abbreviations, such as "configuration" which is abbreviated as "config", "conf", or even "cfg". At the same time the abbreviation "pos" can signify "position" or "positive". We included all the options we found in the list.

The second class was acronyms, such as "sql", "jdk", or "pid". We did not expand these, as their use is very common among developers (at least in areas where they are relevant), and they are not thought of as abbreviations but as independent terms in their own right. The third was terms which we did not identify, such as "seps", or that are not really abbreviations,

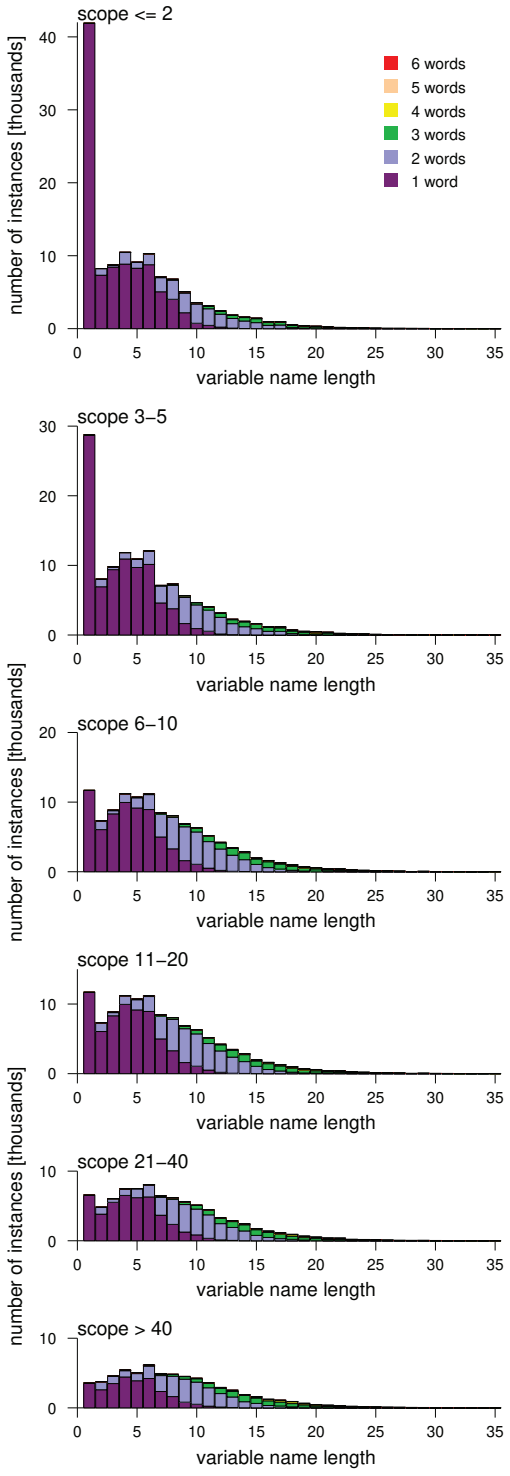

Fig. 5. *Histograms of name lengths and their division into words for the different scope groups.*

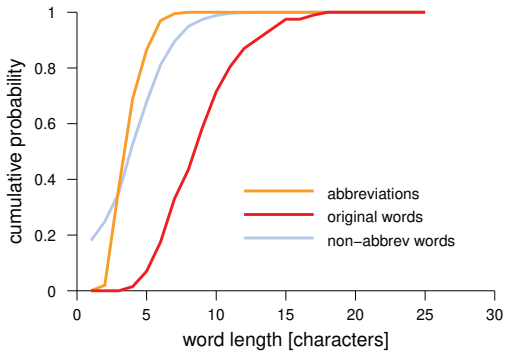

Fig. 6. *CDFs of the lengths of abbreviations and the words which they replace. The CDF of non-abbreviated words is shown for comparison.*

such as "clazz". We did not use these two classes in our analysis of abbreviations.

While looking at the data we observed a few abbreviations that seemed to be missing from the list. We therefore scanned the first 10,000 names, and identified 48 additional abbreviations for which we could suggest the original full words from which they were derived. Altogether we ended up with 199 word-abbreviation pairs. Of these, 18 were actually abbreviations of two words (for example, "regexp" which is an abbreviation of "regular expression"). Some abbreviations appeared both in singular and in plural (indicated by an "s" at the end of the abbreviations, as in "apps" for "applications").

Figure 6 shows the distributions of lengths of these abbreviations and original full words. The distribution of abbreviation lengths is very narrow: most of them are 3–6 letters long. The distribution of word lengths is slightly wider, and most are in the range of 6–13 letters. This largely agrees with the observation made by Aman et al. Their speculation that programmers tend to abbreviate long words is further supported by the distribution of non-abbreviated words. This is similar to the distribution of abbreviations, but slightly wider; the vast majority of such words are 1–8 letters long and only 5% are longer.

D. Popular Names and Words

Apart from abbreviations, it is also interesting to look at the most popular names and the most popular words that are used in names. The top 50 of each are listed in Table II (for names) and Table III (for words). In each table, the first column represents the complete dataset, and the next six are for the different scope groups. The lists of most popular names are similar but not identical to such lists that have been collected in other studies [17], [3].

Two things that can be readily observed concern the variability of the results and the problems of making any concrete statements. First, we see that there are significant differences between the scope groups. Consequently the data for the complete dataset is a sort of weighted average of these groups,

TABLE II
Top names overall and in each of the scope groups.

		By scope					
Rank	*All*	*≤2*	*3–5*	*6–10*	*11–20*	*21–40*	*>40*
1	e	e	e	i	i	i	i
2	i	i	i	result	result	result	result
3	result	ex	result	e	sb	n	type
4	j	result	j	j	c	c	c
5	c	t	c	c	n	index	count
6	value	j	ex	n	j	count	index
7	n	value	s	value	count	sb	n
8	s	ignored	entry	sb	e	builder	name
9	ex	cptr	value	index	list	name	x
10	t	s	intent	count	name	value	action
11	entry	b	r	list	value	start	start
12	index	listener	n	entry	index	j	width
13	sb	c	t	s	size	type	p
14	count	m	index	key	builder	file	height
15	key	intent	m	listener	file	x	sb
16	name	entry	key	r	entry	a	id
17	m	exception	list	name	x	y	a
18	list	f	v	p	s	s	position
19	b	n	p	size	start	m	size
20	r	v	name	child	key	list	builder
21	x	key	count	builder	a	map	response
22	intent	te	args	len	map	width	logger
23	p	field	x	a	type	key	buf
24	a	view	sb	x	p	params	y
25	v	msg	b	m	path	length	v_1
26	builder	index	len	t	m	size	length
27	view	a	item	b	b	id	s
28	size	x	params	params	length	entry	r
29	f	p	view	v	t	r	out
30	file	name	a	file	id	path	value
31	params	r	child	intent	r	url	j
32	type	data	size	length	view	end	data
33	id	ignore	f	data	y	height	line
34	child	item	info	node	width	view	view
35	data	message	file	id	child	position	res
36	listener	builder	node	view	v	p	params
37	start	container	builder	f	params	data	b
38	item	params	res	path	height	t	list
39	len	fields	ret	item	node	e	done
40	y	l	id	info	data	b	in
41	length	file	k	start	end	line	devicesession
42	path	info	data	l	len	offset	key
43	node	list	type	buffer	url	res	tag
44	info	ioe	temp	response	in	v	url
45	res	temp	l	k	out	response	v_2
46	response	__functionaddress	val	map	f	len	m
47	map	id	path	position	offset	request	f
48	field	response	d	in	first	in	v
49	position	d	length	type	response	f	file
50	l	val	array	y	res	buffer	t

but is not very similar to any of them. Second, is seems that at least part of the variability is the result of large projects which contain very repetitive naming. This allows some names that are not really common in general to reach the top-50 lists; examples include __functionAddress and deviceSession.

Digging into the data, another interesting observation is that there are names with scope-dependent behavior, while others do not exhibit such behavior. For example, the second and third most popular names in the whole dataset are i and result. These names are also found at ranks 1 to 4 in all of the scope groups, so they are indeed universally popular. But the top-ranked name overall is e, which is commonly used for variables representing a thrown exception. This is the top-ranked name up to a scope of 5 lines, the third most popular at scopes of 6–10, the eighth for 11–20, the 39th for 21–40, and doesn't make it into the top 50 for broad scopes of more than 40 lines (it is at rank 61). The code blocks handling exceptions are apparently usually quite short.

Another interesting observation is the prevalence of single-letter names. It is not just i and e. Many other letters appear. In the full dataset, 17 of the 50 top names are single-letter names. In the different scope groups it is between 15 and 18 names. 13 of these letters appeared in the top 50 lists of *all* the groups, and also of the whole dataset. These are a,

TABLE III
Top words (standalone and part of compound names) overall and in each of the scope groups.

		By scope					
Rank	*All*	*≤2*	*3–5*	*6–10*	*11–20*	*21–40*	*>40*
1	e	e	e	i	i	i	name
2	i	i	i	name	name	name	type
3	name	result	result	result	result	type	2
4	result	ex	name	list	file	file	i
5	index	value	value	e	type	result	id
6	value	c	view	index	id	index	1
7	id	t	index	value	index	id	index
8	type	name	id	id	list	count	count
9	file	j	m	count	count	list	x
10	list	view	key	view	view	size	v
11	view	key	list	size	value	x	start
12	c	new	c	file	size	start	size
13	count	index	type	key	x	view	y
14	m	m	s	c	2	y	width
15	2	ignored	j	type	1	1	file
16	key	s	file	m	c	2	list
17	x	b	new	data	start	m	m
18	size	ptr	entry	new	data	width	data
19	1	listener	info	2	m	data	result
20	data	intent	ex	n	key	is	is
21	new	id	x	1	y	value	height
22	start	file	r	j	n	height	view
23	n	2	intent	x	path	c	f
24	y	type	count	child	new	n	max
25	j	data	2	entry	is	key	offset
26	s	entry	size	info	sb	field	a
27	info	exception	item	r	height	map	c
28	t	f	t	node	map	path	end
29	path	field	n	path	width	end	value
30	v	n	data	s	info	new	time
31	item	v	1	builder	builder	class	path
32	r	list	child	map	child	builder	b
33	is	item	path	item	class	current	p
34	entry	info	node	start	node	length	to
35	child	1	p	p	offset	offset	current
36	width	x	v	y	p	num	num
37	height	a	b	sb	item	info	n
38	p	p	y	is	to	max	last
39	node	msg	a	listener	e	last	text
40	b	te	time	class	s	to	key
41	map	r	params	length	max	time	info
42	a	path	start	width	r	item	in
43	field	node	array	t	entry	r	new
44	builder	method	args	height	current	child	line
45	f	message	is	v	length	node	field
46	intent	y	len	time	end	url	length
47	ex	builder	map	a	j	request	map
48	class	size	field	array	a	f	item
49	time	params	class	position	in	position	pos
50	offset	child	builder	current	field	a	class

b, c, f, i, j, n, p, r, s, t, v, and x. Their prevalence indicates that they are widely used, and probably reflect some common naming conventions (called "programmer slang" in [3]). In particular, single-letter names are not just loop indexes, and indeed some of them correspond to the more meaningful single-letter names identified by Beniamini et al. [6]. A similar result concerning the use of many different single-letter names has been observed by Gresta et al. [17]. The fact that this does not reflect just random use of short names is bolstered by the observation that seven letters did not appear in *any* of our lists; these are g, h, o, q, u, w, and z.

Similar observations can be made on the lists of most popular words. In particular we note that many single letters appear in all scope groups. This is also true of the broad scope group, and contradicts the results as reported by Aman et al., who claim that only i and v appear in the top 50 terms of this group, and that they are exceptions. But note that our definition of this group is not identical to theirs.

It is also interesting to note words that are used in many different compound names, and therefore achieve a higher ranking than they achieve by themselves. Chief among these are the numbers 1 and 2, which are only used as additives to other names. They come in at ranks 19 and 15 in the full list, and also appear in all the lists of the scope groups, with

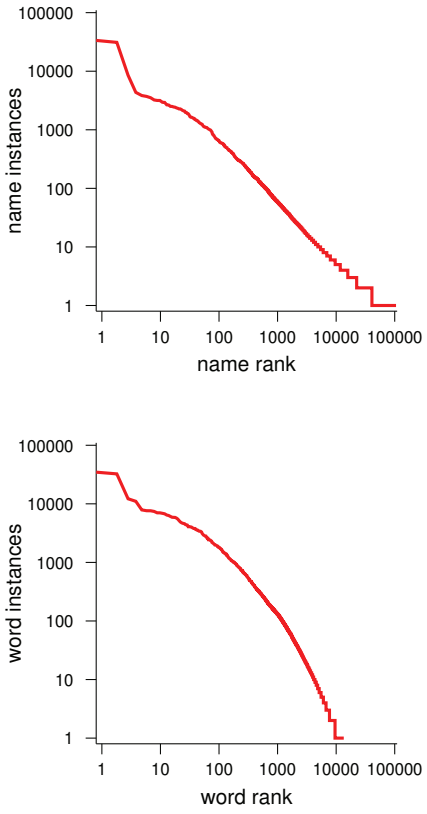

Fig. 7. *Zipf plots of the popularity of names and words.*

somewhat higher ranks in larger scopes. Note that `2` is ranked higher than `1`, perhaps because sometimes the first instance is left unnumbered, or because it is used as shorthand for "to". More generally, Gresta et al. found that a bit more than 6% of names have numbers at their end [17]. In the Aman et al. dataset the total use of numbers appears to be much lower, going up from 0.8% in minimal scopes to 2.6% in broad scopes (Figure 2).

E. Distribution of Popularity

The popularity of names in different ranks is shown in Figure 7. Alltold the dataset contained 105,321 unique names. In the figure they are ranked from the most popular (`e`, which appeared a total of 33,544 times) to the least popular (no less than 64,531 names that appeared only once each). The graph shows the number of appearances as a function of the rank in log-log axes. This rendering leads to a straight line, implying that the popularity of the name at rank k is proportional to $1/k$. This relationship was famously found for the words in natural language texts by linguist George Kingsley Zipf, and is knows as Zipf's law. It has since also been observed in code [11], [3].

Figure 7 also shows a similar graph for the individual words used in names (either as the whole name or as an element of a compound name). In this case the graph is not a straight line. This is a result of the rather limited basic vocabulary being used: the 105,321 unique names in the dataset are composed of only 13,488 unique words. When more names are added, the frequency of words keeps rising, but new ranks (that is, new unique words) are added less frequently, so the graph curves downwards [30], [3].

Returning to full names, it is obvious that the popular ones are much more common than the rare ones. It is therefore interesting to note how dominant the popular names are overall. We quantify this using the $N_{1/2}$ metric, which is the fraction of all the unique names, and specifically the more popular ones, that together account for half of all the name instances in the code [15]. In the full dataset this is 0.43% — less than half of one percent of the names. If we look at the scope groups separately, this grows monotonically from 0.37% in the minimal scope group to 6.73% in the broad scope group. This implies that the popular names are more dominant in narrow scopes, and that the distribution is somewhat less skewed in broader scopes.

IV. Threats to Validity

Our work is based on data collected and processed by Aman et al. [2]. As such it is subject to all the threats to validity of that work. For example, Aman et al. limited their work to code in the Java language, in order to avoid the possible confounding factor of programming languages. This entails a threat to external validity, as the results may not generalize to other languages.

The main processing they performed was to divide compound names into terms, and to classify these terms. The partitioning algorithm they used may not be optimal, and other results may be obtained by using different procedures such as [18], [10], [21]. The classification depends on the dictionary used to identify dictionary words, and on lists of abbreviations that may be incomplete. To mitigate part of this threat, we added abbreviations that appeared to be missing from the original list. A new threat that was introduced concerns the association of abbreviations with full words. As we noted this association is many-to-many, and we cannot know if any specific association was intended.

Another issue concerns the tabulation of names and words by their popularity. As we noted, it may happen that a name is extremely popular in only one project, but that is enough to make it look very popular globally. We considered the alternative of counting the projects in which a name appears rather than and total number of instances in all projects. The result was that 41 of the top 50 names stayed the same, including 16 of the 17 single-letter ones, albeit their order changed somewhat. It therefore seems that this would not cause a fundamental change to the results.

More generally, however, the details of the results depend on the projects from which names are gathered. For example, other studies have found somewhat different lists of popular names, including in the top ranks [17], [3]. But the general

trends appear to be stable, including the prevalence of single-letter names, names that are types, etc. Consequently we believe that the above results and definition of popularity are generally valid despite these threats.

V. Conclusions

Aman et al. set out to study variables with broad syntactic scope, in terms of their attributes (length and classification) and the words from which they are composed. We extend this work by providing a more detailed analysis that compares variables with different scopes, from a minimal scope of up to 2 lines, up to broad scopes of hundreds of lines.

Using six ranges of scopes, rather then singling out only variables with broad scope, allowed us to make more accurate observations regarding the effect of scope on name choice. We replicated Aman et al.'s result that the names of variables with broad scope indeed tend to be longer, and added the observation that this is part of a progression, and that the biggest differences are that variables with minimal scope are shorter. Likewise, we replicated the result that more variables with broad scope have compound names, and less of them have single-letter or dictionary word names. To this we added the observation that the results for compound names and single-letter names are also part of a progression, with single letter names being especially common in narrow scopes. But dictionary word names are different, and have roughly the same prevalence in medium and narrow scopes, which are just slightly higher than in broad scopes.

In addition we also replicated Aman et al.'s observation that long words tend to be abbreviated, and noted that this starts for words of 6 letters. We also looked at the most commonly used names and words, both overall and for different scopes, and found that single letter names are quite popular, and that many different letters are used.

A major shortcoming of this study is that Aman et al. consider only the scopes of local variables within a function or method. They do not compare local variables with data members of the class (fields), which have global scope within the class. Their justification for this omission was that data members are part of the class design, and therefore their names are usually given by designers, not by programmers. Comparing them to names given by programmers writing functions may then introduce a confounding factor of who gave the names.

While this is indeed a valid concern, we believe that a comparison of the names of data members and local variables still has value. When developers think of different scopes, they usually think of the differences between class scope and function scope. The fact that the names may be given by different people is part of this. But the collection of the needed data for such a study is left for future work.

The presented data provides a detailed portrait of the current practice of naming in open source Java projects. This includes a characterization of how local variables' names' lengths and structure change with scope, and the identification of the most popular names in each scope. These results suggest two directions for deeper research:

1) What are the reasons and implications of the different structures used in different scopes? Are longer names with more words indeed needed in order to comprehend longer code segments? In other words, is the use of more characters and words justified? Our detailed results concerning what developers do in practice enable a focused study of the possible benefits of specific behaviors, such as increased use of compound names in larger scopes.
2) The most popular names include many more different single-letter names than expected, which appears to diverge from common naming guidelines. This suggests several followup questions. When exactly are such names used and how does this correlate with scope? As a specific example, why does the popularity of the name `t` plummet with increased scope, while that of `x` grows? Does the use of single-letter names come with a price of reduced comprehensibility? Our results identify specific popular single-letter names, thus providing the motivation and the foundation for future studies of this phenomenon.

Making progress on these further research directions can be expected to lead to improved naming guidelines that will be useful for practitioners.

Experimental Materials

Complete experimental materials and results are available on Zenodo using DOI 10.5281/zenodo.7699870

References

[1] H. Aman, S. Amasaki, T. Sasaki, and M. Kawahara, "*Empirical analysis of change-proneness in methods having local variables with long names and comments*". In *Intl. Symp. Empirical Softw. Eng. & Measurement*, pp. 50–53, Oct 2015, DOI: 10.1109/ESEM.2015.7321197.

[2] H. Aman, S. Amasaki, T. Yokogawa, and M. Kawahara, "*A large-scale investigation of local variable names in Java programs: Is longer name better for broader scope variable?*" In 14th *Quality of Inf. & Commun. Tech.*, pp. 489–500, Sep 2021, DOI: 10.1007/978-3-030-85347-1_35.

[3] N. Amit and D. G. Feitelson, "*The language of programming: On the vocabulary of names*". In 29th *Asia-Pacific Softw. Eng. Conf.*, pp. 21–30, Dec 2022.

[4] V. Arnaoudova, M. Di Penta, and G. Antoniol, "*Linguistic antipatterns: What they are and how developers perceive them*". *Empirical Softw. Eng.* **21(1)**, pp. 104–158, Feb 2016, DOI: 10.1007/s10664-014-9350-8.

[5] E. Avidan and D. G. Feitelson, "*Effects of variable names on comprehension: An empirical study*". In 25th *Intl. Conf. Program Comprehension*, pp. 55–65, May 2017, DOI: 10.1109/ICPC.2017.27.

[6] G. Beniamini, S. Gingichashvili, A. Klein Orbach, and D. G. Feitelson, "*Meaningful identifier names: The case of single-letter variables*". In 25th *Intl. Conf. Program Comprehension*, pp. 45–54, May 2017, DOI: 10.1109/ICPC.2017.18.

[7] D. Binkley and D. Lawrie, "*The impact of vocabulary normalization*". *J. Softw.: Evolution & Process* **27(4)**, pp. 255–273, Apr 2015, DOI: 10.1002/smr.1710.

[8] D. Binkley, D. Lawrie, S. Maex, and C. Morrell, "*Identifier length and limited programmer memory*". *Sci. Comput. Programming* **74(7)**, pp. 430–445, May 2009, DOI: 10.1016/j.scico.2009.02.006.

[9] B. Caprile and P. Tonella, "*Restructuring program identifier names*". In *Intl. Conf. Softw. Maintenance*, pp. 97–107, Oct 2000, DOI: 10.1109/ICSM.2000.883022.

[10] N. R. Carvalho, J. J. Almeida, P. R. Henriques, and M. J. Varanda, "*From source code identifiers to natural language terms*". *J. Syst. & Softw.* **100**, pp. 117–128, Feb 2015, DOI: 10.1016/j.jss.2014.10.013.

[11] C. Casalnuovo, K. Sagae, and P. Devanbu, "*Studying the difference between natural and programming language corpora*". *Empirical Softw. Eng.* **24(4)**, pp. 1823–1868, Aug 2019, DOI: 10.1007/s10664-018-9669-7.

[12] F. Deissenboeck and M. Pizka, "*Concise and consistent naming*". *Softw. Quality J.* **14(3)**, pp. 261–282, Sep 2006, DOI: 10.1007/s11219-006-9219-1.

[13] A. Etgar, R. Friedman, S. Haiman, D. Perez, and D. G. Feitelson, "*The effect of information content and length on name recollection*". In 30th *Intl. Conf. Program Comprehension*, pp. 141–151, May 2022, DOI: 10.1145/3524610.3529159.

[14] S. Fakhoury, D. Roy, Y. Ma, V. Arnaoudova, and O. Adesope, "*Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization*". *Empirical Softw. Eng.* **25(3)**, pp. 2140–2178, May 2020, DOI: 10.1007/s10664-019-09751-4.

[15] D. G. Feitelson, "*Metrics for mass-count disparity*". In 14th *Modeling, Analysis & Simulation of Comput. & Telecomm. Syst.*, pp. 61–68, Sep 2006, DOI: 10.1109/MASCOTS.2006.30.

[16] D. G. Feitelson, A. Mizrahi, N. Noy, A. Ben Shabat, O. Eliyahu, and R. Sheffer, "*How developers choose names*". *IEEE Trans. Softw. Eng.* **48(1)**, pp. 37–52, Jan 2022, DOI: 10.1109/TSE.2020.2976920.

[17] R. Gresta, V. Durelli, and E. Cirilo, "*Naming practices in Java projects: An empirical study*". In XXth *Brazilian Symp. Softw. Quality*, art. 10, Nov 2021, DOI: 10.1145/3493244.3493258.

[18] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "*An empirical study of identifier splitting techniques*". *Empirical Softw. Eng.* **19(6)**, pp. 1754–1780, Dec 2014, DOI: 10.1007/s10664-013-9261-0.

[19] J. C. Hofmeister, J. Siegmund, and D. V. Holt, "*Shorter identifier names take longer to comprehend*". *Empirical Softw. Eng.* **24(1)**, pp. 417–443, Feb 2019, DOI: 10.1007/s10664-018-9621-x.

[20] G. J. Holzmann, "*Code clarity*". *IEEE Softw.* **33(2)**, pp. 22–25, Mar/Apr 2016, DOI: 10.1109/MS.2016.44.

[21] M. Hucka, "*Spiral: Splitters for identifiers in source code files*". *J. Open Source Softw.* **3(24)**, art. 653, 2018, DOI: 10.21105/joss.00653.

[22] B. W. Kernighan and R. Pike, *The Practice of Programming*. Addison-Wesley, 1999.

[23] D. Lawrie, H. Feild, and D. Binkley, "*Quantifying identifier quality: An analysis of trends*". *Empirical Softw. Eng.* **12(4)**, pp. 359–388, Aug 2007, DOI: 10.1007/s10664-006-9032-2.

[24] D. Lawrie, C. Morrell, H. Field, and D. Binkley, "*Effective identifier names for comprehension and memory*". *Innovations in Syst. & Softw. Eng.* **3(4)**, pp. 303–318, Dec 2007, DOI: 10.1007/s11334-007-0031-2.

[25] O. A. L. Lemos, M. Suzuki, A. C. de Paula, and C. Le Goes, "*Comparing identifiers and comments in engineered and non-engineered code: A large-scale empirical study*". In 35th *ACM Symp. Applied Computing*, pp. 100–109, Mar 2020, DOI: 10.1145/3341105.3373972.

[26] B. Liblit, A. Begel, and E. Sweetser, "*Cognitive perspectives on the role of naming in computer programs*". In 18th *Psychology of Programming Workshop*, pp. 53–67, Sep 2006.

[27] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftmanship*. Prentice Hall, 2009.

[28] G. Scanniello, M. Risi, P. Tramontana, and S. Romano, "*Fixing faults in C and Java source code: Abbreviated vs. full-word identifier names*". *ACM Trans. Softw. Eng. & Methodology* **26(2)**, art. 6, Oct 2017, DOI: 10.1145/3104029.

[29] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, and M. Beigl, "*Descriptive compound identifier names improve source code comprehension*". In 26th *Intl. Conf. Program Comprehension*, pp. 31–40, May 2018, DOI: 10.1145/3196321.3196332.

[30] O. Tripp and D. G. Feitelson, *Zipf's Law Revisited*. Tech. Rep. 2007-115, Hebrew University, Aug 2007.