Scientific Research Publishing

Journal of Software Engineering and Applications, 2021, 14, 1-10
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Linguistic Economy Applied to Programming Language Identifiers

Michael A. Dorin[1,2*], Sergio Montenegro[2]

[1]University of St. Thomas, St. Paul, USA
[2]Universität Würzburg, Würzburg, Germany
Email: *mike.dorin@stthomas.edu

How to cite this paper: Dorin, M.A. and Montenegro, S. (2021) Linguistic Economy Applied to Programming Language Identifiers. *Journal of Software Engineering and Applications*, **14**, 1-10.
https://doi.org/10.4236/jsea.2021.141001

Received: October 16, 2020
Accepted: January 9, 2021
Published: January 12, 2021

Abstract

Though many different readability metrics have been created, there still is no universal agreement defining readability of software source code. The lack of a clear agreement of source code readability has ramifications in many areas of the software development life-cycle, not least of which being software maintainability. We propose a measurement based on Linguistic Economy to bridge the gap between mathematical and behavioral aspects. Linguistic Economy describes efficiencies of speech and is generally applied to natural languages. In our study, we create a large corpus of words that are likely to be found in a programmer's vocabulary, and a corpus of existing identifiers found in a collection of open-source projects. We perform a usage analysis to create a database from both of these corpora. Linguistic Economy suggests that words requiring less effort to speak are used more often than words requiring more effort. This concept is applied to measure how difficult program identifiers are to understand by extracting them from the program source and comparing their usage to the database. Through this process, we can identify source code that programmers find difficult to review. We validate our work using data from a survey where programmers identified unpleasant to review source files. The results indicate that source files identified as unpleasant to review source code have more linguistically complicated identifiers than pleasant programs.

Keywords

Corpus, Programming Languages, Comprehension, Complicated

1. Introduction

Even though many guidelines exist for identifier creation, there is no specific

metric to measure the overall readability of identifiers. Selecting understandable identifiers is a crucial part of program maintenance. In the field of linguistics, semantics is the area that focuses on meaning which is crucial to understand. Identifiers represent source code elements such as variables and method names, and are chosen by a human programmer during software development. Since up to 70% of source code is made up of identifiers [1], they are a critical component of program semantics.

Zipf's "law" of vocabulary balance was chosen as the basis for measuring understandably. George Zipf's "law" of vocabulary balance predicts an orderly distribution between word size and word usage in spoken languages. Zipf described how words used in speech were subject to two opposing forces. One force from the speaker endeavors to economize efforts by reducing their vocabulary size to have the smallest vocabulary necessary to convey all meanings. Opposing this force is the need for precise communication, which encourages an ever-increasing vocabulary until there is a word for every meaning [2]. The result is that words that are easier to speak and understand are used more often. The work Économie des changements phonétiques by André Martinet, supports Zipf's ideas by describing these opposing forces of communication needs and the natural human tendency of wanting stability, as Linguistic Economy. [3].

While analyzing word usage from James Joyce's novel *Ulysses*, Zipf [2] noticed word rank (r) multiplied by the frequency of use (f) yielded a constant (C) (Equation (1)). For example, the tenth ranked word in *Ulysses* was used 2653 times in the book, giving a constant of 26,530. The 20th ranked word multiplied by its frequency of 1311 gave a constant of 26,220. Notice the resulting constants from both examples are very close in value. We can show this as an equation below:

$$r \times f \approx C \tag{1}$$

Using algebra, the equation can be modified as follows in Equation (2):

$$f \approx \frac{C}{r} \tag{2}$$

When Zipf plotted the rank of words from *Ulysses* against frequency of use on a double logarithmic scale, the result was a line with slope of negative one. Since Zipf's plots are logarithmic, we take the logarithm on both sides and adjust our equation to see Zipf's equation in a different light:

$$\log(f) \approx \log(C) - \log(r) \tag{3}$$

Equation (3) demonstrates Zipf's equation, when plotted on a double logarithmic scale, with a slope of negative one. For the purpose of consistent terminology in this paper, going forward, log(f) will be referred to as Zipf Frequency. That is to say, the Zipf Frequency is the logarithm of the frequency of use.

Reviewing existing literature, it is apparent there is no universal agreement on the correctness of Linguistic Economy. Duarte describes how the concept has been studied and reviewed by many scholars, and their conclusions are not all

supportive [4]. Even in 1965, George Miller wrote in the introduction to the reprint of Zipf's "The Psycho-Biology of Language: An Introduction to Dynamic Philology" that a collection of monkeys with typewriters would get the same word size and frequency distribution as indicated by the Law of Vocabulary Balance [5]. Duarte seemed to focus on the concept of Linguistic Economy as an application of laziness [4]. However, a close reading of Zipf and Martinet shows the focus is not on laziness, but on efficiency, which is very applicable to software development. We must also remember that even Zipf described this phenomenon as an observational study. With this caveat in mind, we believe it is not unreasonable to recognize patterns of Linguistic Economy in the work of humans. Yu *et al.* are supportive of Zipf, and Martinet has shown the applicability of Zipf's Law in fifty different languages [6]. Zipf's Law and software have also been explored, and the work by Zhang shows that Zipf's law can be applied to improve to the classic software length metric as originally developed by Halstead in 1977 [7] [8]. This initial usage of Zipf's law to improve existing metrics is very promising, and shows additional research is warranted to further establish how the law can be used to connect human behavioral and mathematical aspects of programming.

Although the work of Zipf has been applied in the area of software, Linguistic Economy has not sufficiently been leveraged to measure program readability. Therefore we investigate how Zipf Frequencies can be applied to measure the complicacy of identifiers found in program source code. From a natural language perspective, if an identifier is composed of intricate and uncommon words, it will be more difficult for a programmer to comprehend. From a programming language perspective, rarely used identifiers will be harder for a programmer to understand.

To create a metric to measure identifier quality, we first build necessary corpus databases using vocabulary and identifiers programmers would likely be familiar with. We then compared data from an earlier survey where programmers were asked to rate the desirability of a program source file for review. The results show that our methodology can give a metric for identifier readability. Using methods from linguistic economy, it is possible to review and evaluate the quality of identifiers.

2. Methods and Data

2.1. The Five Finger Rule

In season three, episode twelve of the American sitcom *Seinfeld*, the character Elaine helped the character George land a job at a publishing company. As a "thank you" present, George buys Elaine a cashmere sweater that has a minor flaw, a red dot, for which the price was reduced considerably. George believes she will not notice the spot, but when Elaine becomes aware of it, she becomes angry and returns the sweater [9]. Humans seem to have a negativity bias, giving greater importance to negatives rather than positives [10]. In the *Seinfeld* epi-

sode, Elaine was unable to overlook the tiny flaw in an otherwise beautiful sweater. In software engineering, we can overlook the quality of the nicely written parts of the source code because of the difficult and complicated parts.

Another way of approaching negativity bias lies in when a person would give up reading a book. Research in the field of foreign language acquisition suggests what is known as the "Five Finger Rule" for text difficulty can apply to second languages. The rule states as you read text, each time you encounter a word too difficult for you, you lift a finger. If you lift five fingers before completing a page, the text is likely too difficult for you [11] [12]. Irrespective of how many words are understandable, a few incomprehensible words make text more difficult for readers. We apply these concepts to software readability.

2.2. Existing Data: Programmer Survey

In preparation for XP'18: The 19th International Conference on Agile Software Development, a survey was performed in an attempt to determine if we could identify aspects of source code that programmers find unpleasant [13]. In this survey ninety, C++ source files were made available for evaluation on a public website. Volunteers were presented source code files and asked if they would be pleasant or unpleasant to review. More than 400 people participated in the survey.

The original research for XP'18 was focused on the ascetic nature of the source code with no consideration of language syntax and semantics [14]. A summary of the results is shown in **Table 1**. In order to qualify files for further review, a ratio was made between pleasant to review and unpleasant to review results. For example, if a file had 20 respondents state it would be pleasant to review and 15 state it would be unpleasant to review, the result would be 1.3. For this paper, we considered files with a ratio of less than 1.0 leaning unpleasant to review and those was a ratio greater than or equal to 1.0 leaning pleasant to review.

2.3. Natural Language Corpus Creation

In order to evaluate the words that are put together to make an identifier, a natural language corpus was necessary to perform measurements against. Though it is not difficult to find a corpus of word and usage frequency data for English, we required a collection that included domain-specific words that programmers would likely have encountered in their professional careers. This goal was achieved by using words found in technology and programming related internet Usenet newsgroups. Though officially discontinued now, in the past, Usenet

Table 1. Pleasant vs. unpleasant to review.

Total C++ Files	**91**
Leaning pleasant to Review Files	41
Leaning unpleasant to Review Files	50

newsgroups were a popular way of exchanging information [15]. We selected Usenet newsgroups as they provide a wealth of conversations among people with technical backgrounds. We wrote a small program to separate words, with very rudimentary filtering applied to remove things like routing information. The resulting corpus made up of the words found in these technical newsgroups demonstrates a Zipf distribution. When plotted as shown in Figure 1 the line is very close to having a slope of negative one. The most popular words found in the corpus are shown in Table 2. The resulting corpus contained 1,228,823 entries. As a final note, no attempt was made to remove words considered spam from the corpus, and the corpus includes many invented words designed to trick spam filters.

2.4. Existing Identifier Corpus Creation

For the corpus based on existing code identifiers, we downloaded more than eight hundred open-source projects found on GitHub [16]. Identifiers were then extracted from all the C and C++ files, and the corpus was compiled with their usage frequency data. This corpus also demonstrates a Zipf equation with a slope very close to negative one as shown in Figure 2. Many of the most popular identifiers happen to be very short. See Table 3 for a breakdown of popular identifiers. The resulting corpus has 8,420,238 entries.

2.5. Source Code Analysis

For pleasant and unpleasant program files presented in the survey, we extracted the identifiers. For the natural language measurements, we further segmented identifiers into component words. For example, a camelCase formatted identifier would be segmented into "camel" and "case," as that is how a human might

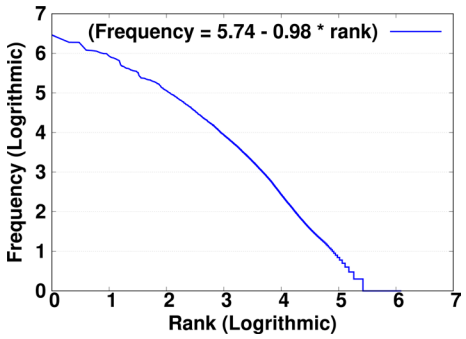

Figure 1. Usenet words rank frequency distribution.

Table 2. Most popular words.

Word	Count
the	2,921,314
to	1,930,521
a	1,920,627
is	1,214,419

Table 3. Most popular identifiers.

Identifier	Count
i	4,976,942
std	4,085,864
NULL	1,946,649
T	1,645,339
p	1,615,373
x	1,597,532
type	1,581,043
s	1,426,045
size	1,318,116
a	1,308,500
value	1,256,133
size_t	1,161,173
data	1,039,361
c	1,014,945
b	934,781
h	932,624
n	927,145
j	903,268
y	895,105

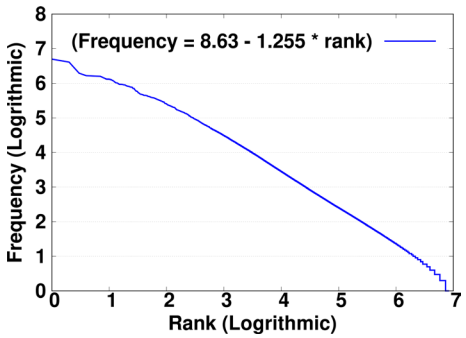

Figure 2. Program identifiers rank frequency distribution.

read them. As negativity bias and the "Five Finger Rule" play a strong role in deciding what is pleasant or unpleasant, we selected a collection of the five least popular component words from each source file considered leaning pleasant to review and leaning unpleasant to review. At this point, popularity of the selected component words was looked up in the natural language corpus and tallied.

The second step was to compare the identifiers found in source code from the survey to a corpus we made of identifiers from large projects found on GitHub [16]. In this analysis, we did not segment the component words of the identifier; we used the entire identifier when searching the identifier database. As before,

we considered negativity bias and the "Five Finger Rule," and selected five of the least popular identifiers from each leaning pleasant to review and the leaning unpleasant to review source file. As before, the popularity of the selected identifiers was found and scores were tallied.

3. Results

Human Language Corpora

The Usenet corpora results results are shown in Table 4. Programs which unpleasant to review, as reflected by the median frequency of use, have identifiers created from more complicated natural language words. For example, considering the Five Finger Rule, pleasant to review files have a median Zipf frequency of 1.87, which was about 25% higher than unpleasant to review files. Considering all the identifiers, pleasant to review files have a median Zipf frequency of 4.05, which is still higher than unpleasant to review files. Also, an interesting result, unpleasant to review program files, have more words missing from the corpus, as shown in Table 4.

The Five Finger Rule evaluation the corpus based on actual program identifiers. Files that were pleasant to review have a median frequency use of 1.4, which was much higher than the median for unpleasant to review files. Considering all the identifers, the median Zipf frequency of pleasant to review files is 5.6, which is considerably better than unpleasant to review files. As before, unpleasant to review files had more identifiers that were missing from the corpus. This can be seen in Table 5.

Table 4. Results based on Usenet corpus.

Five Finger Rule and Usenet	Value
Total unique words	455
Not included words from unpleasant files	36
Not included words from pleasant files	4
Unique words from pleasant files	205
Unique words from unpleasant files	250
Median Zipf frequency of pleasant Files*	1.86
Median Zipf frequency of unpleasant Files*	1.36
All Usenet Words	**Value**
Total unique words	11,543
Not included words from unpleasant files	36
Not included words from pleasant files	4
Unique words from pleasant files	4052
Unique words from unpleasant files	7491
Median Zipf frequency of pleasant Files*	4.05
Median Zipf frequency of unpleasant Files*	3.87

Table 5. Results based on actual project identifiers.

Five Finger Rule Identifiers	Count
Total ids	455
Missing from Pleasant Files	3
Missing from Unpleasant Files	163
Median Zipf frequency for pleasant files*	1.4
Median Zipf frequency for unpleasant files*	0.15
All Identifiers	**count**
Total ids	99,365
Missing from Pleasant Files	1609
Missing from Unpleasant Files	3534
Mean Zipf frequency for pleasant files*	5.6
Mean Zipf frequency for unpleasant files*	0.75

Table 6. Results based on book based corpus.

Five Finger Rule Reasonableness	Count
Total unique words	455
Unique words from pleasant files	205
Unique words from unpleasant files	250
Not included words from Pleasant Files	74
Not included words from Unpleasant Files	200
Median Zipf frequency of pleasant files*	1.25
Median Zipf frequency of unpleasant files*	0.62
All Words from Reasonableness	**Count**
Total unique words	11,543
Not included words from unpleasant files	274
Not included words from pleasant files	74
Unique words from pleasant files	4052
Unique words from unpleasant files	7491
Median Zipf frequency of unpleasant Files*	3.08
Median Zipf frequency of pleasant Files*	3.26

*Higher is better.

4. Discussion

The results do reflect a connection between the Zipf Frequency of an identifier and desirability of review source code where that identifier is found. This analysis used two different styles of corpus, one common words found in a programmers vocabulary, the second actual identifiers found in open source projects. To gain additional confidence in the results, a "reasonableness check" was run using an additional, but much smaller corpus (91,948 entries) created from open-source

computer books, science books, and miscellaneous novels. The same test was run and the results are summarized in Table 6. A difference between pleasant to review and unpleasant to review files is still reflected with this corpus with pleasant to review files having a median frequency of use higher than unpleasant to review files.

As an additional confidence check, we reran the overall analysis increasing the number of identifiers measured to 100% from each file (rather than the least popular five). Even with the new numbers, files classified as pleasant to review still had an advantage over files classified as unpleasant to review. For example, evaluating using the Usenet corpus, the median of the pleasant to review files was 4.05. while the median of the unpleasant files was 3.87. When evaluating the program identifiers corpus, the value difference was much pronounced. Files classified as pleasant to review had a Zipf frequency median of 5.6, while files classified as unpleasant to review had a Zipf frequency mean of 0.75.

5. Conclusion

Since identifiers play such a prominent part of program source, it is important that they are readable and understandable. A tendency does exist for more complicated identifiers to make a program less desirable to review. In our study, we demonstrated that the concept of Linguistic Economy can be applied to programming, and specifically applied to program readability. Using the results from the survey, asking programmers to identify unpleasant code demonstrates that the words used in human-created identifiers impact the desirability of review. The survey analyses and also shows that actual identifiers themselves, when measured against our identifier corpus, impact the desirability of review. The differences between pleasant and unpleasant to review are files much more pronounced in the actual identifier corpus. There seems to be a significant impact on identifier names that programmers do not see frequently. This demonstrates a connection between the Linguistic Economy of identifiers and program readability. This knowledge can be employed to make more readable, higher quality programs.

Acknowledgements

Sincere thanks to Patrick Dorin and Anders Koskinin for their efforts editing and reviewing this work.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Deissenboeck, F. and Pizka, M. (2006) Concise and Consistent Naming. *Soft-Ware Quality Journal*, **14**, 261-282. https://doi.org/10.1007/s11219-006-9219-1

[2] Zipf, G.K. (1949) Human Behavior and the Principle of Least Effort. Addison-Wesley Press, Boston.

[3] Martinet, A. (1955) Économie des changements phonétiques. *Francke*, **396**, 2-15.

[4] Duarte, M.J.P. (2007) El principio de economía lingüística. *Pragma-Lingüística*, **15-16**, 166-178.

[5] Zipf, G.K. (1936) The Psychobiology of Language: An Introduction to Dynamic Philology. Houghton-Mifflin, Boston.

[6] Yu, S., Xu, C. and Liu, H. (2018) Zipf's Law in 50 Languages: Its Structural Pattern, linguistic Interpretation, and Cognitive Motivation. https://arxiv.org/abs/1807.01855

[7] Zhang, H. (2008) Exploring Regularity in Source Code: Software Science and Zipf's Law. 15*th Working Conference on Reverse Engineering*, Antwerp, 15-18 October 2008, 101-110. https://doi.org/10.1109/WCRE.2008.37

[8] Halstead, M.H., *et al.* (1977) Elements of Software Science. Elsevier, New York.

[9] David, L. and Seinfeld, J. (1991) Seinfeld the Red Dot Episode.

[10] Rozin, P. and Royzman, E.B. (2001) Negativity Bias, Negativity Dominance, and Contagion. *Personality and Social Psychology Review*, **5**, 296-320. https://doi.org/10.1207/S15327957PSPR0504_2

[11] Darby, M., *et al.* (2008) Library Reading Program for Secondary esl Students. *Access*, **22**, 9.

[12] Bryan, S. (2011) Extensive Reading, Narrow Reading and Second Language Learners: Implications for Libraries. *The Australian Library Journal*, **60**, 113-122. https://doi.org/10.1080/00049670.2011.10722583

[13] Dorin, M. (2018) Coding for Inspections and Reviews. *Proceedings of the* 19*th International Conference on Agile Software Development*: *Companion*, Porto, May 2018, 1-3. https://doi.org/10.1145/3234152.3234159

[14] Dorin, M. and Montenegro, S. (2019) Eliminating Software Caused Mission Failures. 2019 *IEEE Aerospace Conference*, Big Sky, 2-9 March 2019, 1-4. https://doi.org/10.1109/AERO.2019.8741837

[15] Hauben, M. and Hauben, R. (1998) Netizens: On the History and Impact of Usenet and the Internet. *First Monday*, **3**. https://doi.org/10.5210/fm.v3i7.605 https://journals.uic.edu/ojs/index.php/fm/ article/view/605

[16] GitHub. http://www.github.com