An Empirical Exploration of Regularities in Open-Source Software Lexicons

Derrin Pierret and Denys Poshyvanyk
Computer Science Department
The College of William and Mary
Williamsburg, VA 23185
{dpierret, denys}@cs.wm.edu

Abstract

The software lexicon is an important source of information during program comprehension activities and it has been in the focus of several recent case studies. Identifiers and comments, which constitute a lexicon in software, encode domain concepts and design decisions made by programmers.

The paper presents an exploratory study that investigates regularities in the software lexicons of open-source projects by analyzing distributions of tokens in diverse software artifacts. The study examined source code of 142 systems from different domains, written in 12 different programming languages, as well as bug reports and external documentation. We discover that distributions of lexical tokens in studied artifacts follow the Zipf-Mandelbrot law, which is an empirical law in statistical natural language processing. Furthermore, the study reveals that the Zipf-Mandelbrot law is not confined to program lexicons in object-oriented languages, as shown in the previous studies, but also emerges in source code written using procedural, functional and markup languages, as well as other software artifacts.

Our study also indicates that a previously devised software science equation does not hold for describing the program vocabulary–length relationship and more studies are necessary.

1. Introduction

Identifiers and comments, which constitute the software lexicon, play an important role in program comprehension as they encode domain concepts and design decisions made by programmers [1, 7]. The recent studies of software lexicons focused on various aspects of structure [5, 9, 12], quality [11, 13] and evolution [2] of identifiers and comments. On the macro-level, the software structures and lexicons have also been studied and shown to obey regularities, such as power laws [3, 4, 14, 18].

These findings have important practical applications. Knowing that software lexicons form structures with predictable characteristics allows us to explain earlier empirical findings in program comprehension research. These structures can also be used in current practice and can outline directions for future research. One such application was presented in the study done by Zhang [18], who observed that the distribution of token occurrences in Java programs follow Zipf's law. Given these findings, they refined Halstead's software science length equation [8] to model the relationship between the length of Java programs and their vocabulary. However, to the best of our knowledge, no previous study attempted to investigate the lexicons of software written in different **p**rogramming **l**anguages (PL) as well as other artifacts, such as software documentation and bug reports (see Table 1). It also remains untested whether the revised software science length equation holds for software written in different languages. In quest for the answers, we formulated the following research questions:

RQ1: *Are there any regularities guiding distributions of lexical tokens in software artifacts? Do these distributions differ for software written in different PLs?*

RQ2: *Does the revised software science length equation hold for software systems written in PLs other than Java?*

We address these two research questions by analyzing the lexicons in source code of 142 software systems and 8 other software artifacts, such as bug reports (e.g., *Eclipse*, *gcc*) and external documentation (e.g., *Java SE documentation*).

2. Powers Laws in Software

The concept of power laws has been around for more than a century dating back to Vilfredo Federico Pareto, who described a power law distribution in the 19th century [16]. Twenty years later, Undy Yule

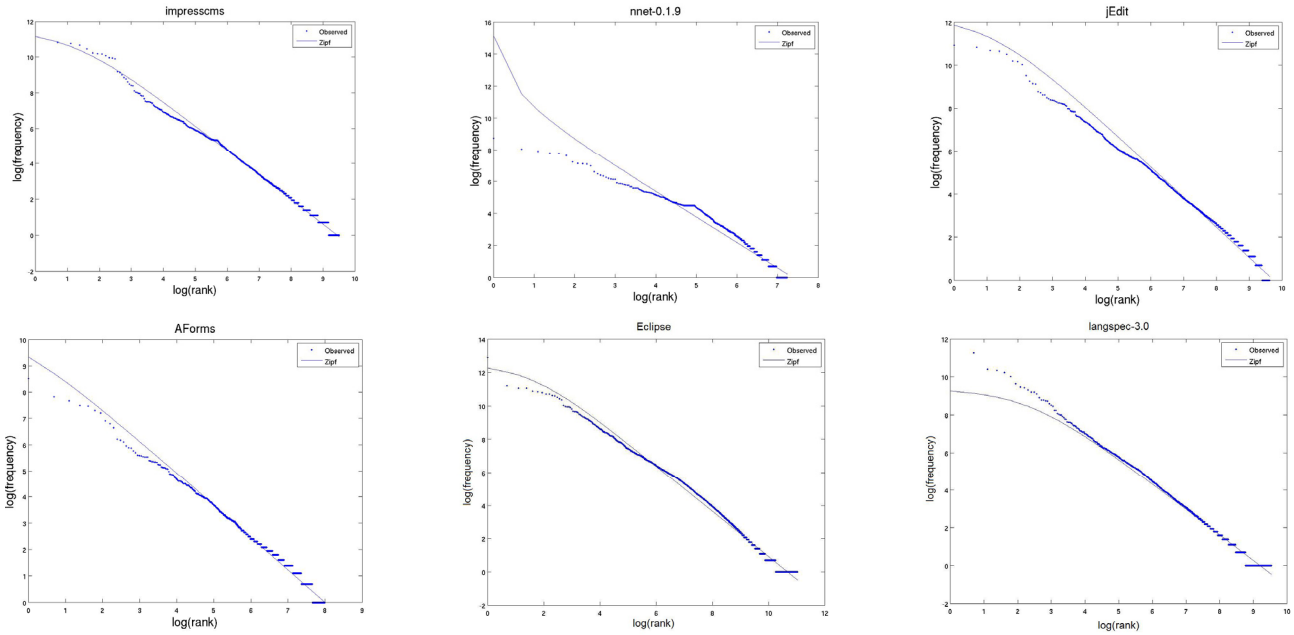

Figure 1. The distributions of token frequencies in four software systems and two software artifacts: *impresscms* (PHP), *nnet* (Matlab), *jEdit* (Java), *AForms* (Haskell), *Eclipse* (bug reports), and the Java SE Documentation. The distribution of the tokens in *impresscms* indicates the best Zipf-Mandelbrot law fit and *nnet* indicates the worst fit from our dataset.

observed a power law distribution in his study of the creation of biological species [17]. About ten years later, George Kingsley Zipf, the linguist from Harvard, discovered an empirical law on word frequencies in natural language speech and texts, which states that while only a few words are used very often, many or most are used rarely [10].

Zipf's law revealed that if each unique word in a body of text is tallied and ranked (from most frequent to least frequent), each word's frequency will be inversely related to that word's rank. The law was generalized with the additions made by Benoit Mandelbrot [15], resulting in the following equation:

$$freq = \frac{C}{(rank + \beta)^{\alpha}} \qquad (1)$$

where *freq* is the frequency of the word, *rank* is the position of the word in the sorted list of words, and α, β, and C are constants. Zipf's law is a special case of the Zipf-Mandelbrot law where $\beta = 0$.

Table 1. Studies of Power Laws in Software

	Analysis		Statistics		Findings
Study	**Text**	**Struc-ture**	**# of Langs**	**# of Systs**	**Power Laws**
Our study	yes	no	12	142	Zipf
Zhang [18]	yes	no	Java	12	Zipf
Concas et al. [4]	no	yes	3	3	Pareto
Baxter et al. [3]	no	yes	Java	56	General
Louridas et al. [7, 14]	no	yes	6	19	General

In terms of mathematics, we are certainly aware of how changing the constants will affect the resulting distribution. What is relatively unknown about these constants (α in particular) is what they mean in terms of the data being analyzed. In attempt to answer this question, a study was performed that compared the Zipf distributions of texts written in English, Spanish, and Russian [6]. The authors found that the constant α was smallest for Russian, was a bit bigger for Spanish, and largest for English, synching up with their respective degrees of inflection, with Russian being most inflective and English being the least inflective. Since the alpha constants seem to have meaning in natural language texts, it is possible that the constants could reflect intrinsic qualities of the source code they are describing.

Other studies confirmed that distributions of lexical tokens and program structures (see Table 1) in source code follow similar patterns as words in natural language documents [18]. More than that, these discovered regularities of lexical tokens in source code led to revising a software science equation for estimating the length of programs (in terms of the total number of tokens) based on their vocabularies (the number of unique tokens). The new *software science equation* for estimating program length was derived using the Zipf-Mandelbrot law as the following [18]:

Table 2. The project statistics, the descriptive statistics on MMRE values for fitting software lexicons to Zipf-Mandelbrot's Law and testing the revised software science equation

	Project statistics				Zipf-Mandelbrot's Law Fit MMREs							Revised soft. science equation MMREs						
Lang/ Artifact	Paradigm	# of Proj	LOC	Voc Size	μ	σ	min	max	25%	Med	75%	μ	σ.	min	max	25%	Med	75%
C++	OOP	41	63K	11,825	0.17	0.04	0.13	0.32	0.15	0.16	0.19	0.80	1.98	0.01	11.35	0.08	0.19	0.37
Java	OOP	10	176K	17,672	0.18	0.03	0.14	0.23	0.17	0.18	0.19	2.23	2.21	0.13	6.57	0.45	1.55	3.24
Smalltalk	OOP	7	3K	1,531	0.17	0.03	0.14	0.22	0.15	0.17	0.18	0.15	0.14	0.03	0.39	0.06	0.11	0.22
C	Proc	10	72K	12,624	0.17	0.03	0.15	0.24	0.15	0.16	0.18	0.16	0.18	0.03	0.64	0.05	0.12	0.19
Matlab	Proc	10	7K	2,251	0.18	0.06	0.12	0.34	0.14	0.16	0.17	1.25	3.48	0.03	11.16	0.05	0.13	0.19
PHP	Proc	10	106K	27,815	0.17	0.05	0.11	0.28	0.15	0.16	0.18	0.95	2.00	0.06	6.57	0.17	0.25	0.36
HTML	Markup	10	0.9K	1,578	0.21	0.05	0.16	0.30	0.17	0.19	0.25	0.22	0.20	0.01	0.68	0.09	0.14	0.25
TeX	Markup	9	2K	2,027	0.15	0.02	0.12	0.19	0.14	0.15	0.17	0.18	0.25	0.01	0.79	0.02	0.09	0.20
XML	Markup	9	23K	16,093	0.19	0.05	0.13	0.26	0.15	0.20	0.21	0.38	0.21	0.14	0.77	0.23	0.28	0.54
Haskell	Func	9	28K	7,083	0.16	0.03	0.13	0.25	0.15	0.15	0.17	0.46	0.56	0.00	1.72	0.09	0.29	0.59
OCaml	Func	8	32K	7,735	0.17	0.04	0.13	0.24	0.15	0.15	0.17	0.95	1.58	0.02	4.31	0.10	0.19	0.80
Scheme	Func	8	35K	6,356	0.16	0.02	0.12	0.18	0.15	0.16	0.17	0.24	0.18	0.01	0.46	0.05	0.29	0.36
Bugs	NL	4	n/a	77,008	0.23	0.08	0.13	0.30	0.19	0.24	0.28	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Docs	NL	5	n/a	68,539	0.21	0.06	0.16	0.30	0.18	0.18	0.24	n/a	n/a	n/a	n/a	n/a	n/a	n/a

$$\hat{N} = C\left(\frac{n^{1-\alpha} - (1+\beta)^{1-\alpha}}{1-\alpha}\right) \quad (2)$$

where N^ is the estimated program length, n is the vocabulary length, and α, β, and C are the constants from the best fit Zipf-Mandelbrot curve for that program.

The power laws can be observed not only in software, but also in everyday life. For example, in the Pareto principle (the law of the vital few), the 80% of the effects come from 20% of the causes.

3. The Exploratory Study

In this exploratory study we investigate if the statistical regularity exhibited by Zipf's law exists in software systems written in various PLs (**RQ1**) as well testing if revised software science length equation holds for software written in PLs other than Java (**RQ2**).

We define a lexical token to be a piece of text in source code that still retains meaning in either spoken language or program understanding. So a token could be an entire camel-case identifier, a word in an underscore separated identifier, or just a semicolon.

We treat a lexical token to be a *word* in a program and the *vocabulary* of a program to be the set of unique words in it. We gather statistics on each token frequency (i.e., the number of occurrences) for each file in each project. For instance, the keyword *public* occurs 327 times on average in C++ programs and only 277 times on average in programs written in OCaml. Then, we sort the tokens in each project according to their frequencies and send them to Matlab, convert the data to a log-log scale and fit the Zipf-Mandelbrot equation to the data. Examples of plotted token frequencies against ranks on a log-log scale are presented in Figure 1.

In order to estimate the goodness of data fit (or accuracy of estimation) into the models described by Zipf's law we use **M**ean **M**agnitude of **R**elative **E**rror (MMRE). Using this measure allows us to compare the results of our study with previous findings [18]. We illustrate the process of computing MMRE values for the *impresscms* project (see Figure 1). The constants generated by fitting the Zipf curve on *impresscms* are α=1.38, β=3 and C=476,889.87. Using the constants, we plug them into the Zipf-Mandelbrot equation to predict the token frequencies. For instance, consider the most frequent token, "–", which occurs 73,216 times in *impresscms*' source files. An MRE of this data point would be the difference between the observed value (73,216) and its predicted value (70,079), divided by the observed value, which in this case results in about 0.044. This number represents the percentage of the observed value that the predicted value is off by, so our predicted value is off by 4.4% of our observed value. After all the token MREs are averaged in the project, the MMRE for each project is computed. We report average MMRE values for all the projects written in the same PL in the Table 2. In addition, we provide a set of descriptive statistical values for MMRE values: the maximum (max), inter-quartile ranges (25% and 75%), median (med), minimum (min), mean (μ) and standard

Table 3. A sample of projects analyzed and the information gathered for each project

Project Description			Zipf-Mandelbrot's Law Fit				Project Stats				Revised soft. science equation		Sourceforge Info	
Proj Name	Lang/ Artifact	Paradigm	α	β	C	MMRE	Avg Token Length	LOC	Voc Size	Proj Size (Tokens)	Est Proj Size	Est MMRE	Devel-opers	Domain
aMule	C++	OOP	1.31	1.00	589,252.14	0.13	9.82	123,830	27,008	1,064,595	1,439,936	0.35	23	File Sharing
datacrow	Java	OOP	1.41	-0.79	450,566.37	0.14	10.27	82,663	10,197	620,399	2,054,546	2.31	1	Organizer
SUnit	Smalltalk	OOP	1.16	0.00	2,048.54	0.15	7.87	2,417	727.00	7,698.00	8,274	0.07	4	Testing
ipcop	C	Proc	1.14	1.00	39,448.52	0.15	7.42	26,463	13,493	200,178	181,268	0.09	38	Network
gpops	Matlab	Proc	1.31	0.00	26,627.79	0.12	6.95	5,586	2,573	88,274	78,463	0.11	5	Math
impresscms	PHP	Proc	1.38	3.00	476,889.87	0.11	9.25	70,782	13,451	741,400	698,712	0.06	53	Internet
Wikipedia	HTML	Markup	1.04	-0.99	1,587.34	0.16	6.85	528	1,612	21,063	18,184	0.14	n/a	Website
CTeX	TeX	Markup	1.20	2.70	15,070.53	0.12	6.57	5,535	3,521	42,772	43,371	0.01	n/a	n/a
PHPXML	XML	Markup	1.64	0.80	104,401.43	0.15	7.17	5,983	1,320	80,227	111,023	0.38	n/a	n/a
lslplus	Haskell	Func	1.17	-0.99	30617.83	0.14	9.26	13467	6892	129,980	354,106	1.72	1	Interpreter
liquidsoap	OCaml	Func	1.44	4.80	284,331.09	0.13	8.23	29,589	6,761	233,358	282,452	0.21	11	Multimedia
nazghul	Scheme	Func	1.46	10.00	724,166.86	0.12	6.93	79,135	11,451	757,828	491,692	0.35	7	Game

deviation (σ). The data is used to provide the overall picture of the differences and similarities among all the MMRE values across projects written in different PLs and paradigms. We provide a sampling of the projects we analyzed in our study in Table 3. The complete data and analysis results for our study are available as an online appendix[1].

3.1 RQ1: Zipf-Mandelbrot Law Evaluation

A commonly acceptable criterion for accurate estimation is MMRE ≤ 0.25 [18]; the lower the MMRE the better. The analysis of source code reveals that lexical token frequency distributions in 132 out of 142 projects can be modeled using the Zipf-Mandelbrot law. Only 10 projects (3 in HTML, 2 each in XML and C++, and 1 each in Matlab, PHP, and Haskell) had MMRE values greater than 0.25.

As we can see in Table 2, in terms of PLs, TeX projects were described best with an MMRE of 0.15, while the worst were HTML systems with MMRE of 0.21. The projects in most languages had an MMRE between 0.16 and 0.18. The results of our study confirm previous findings in [18] that Java projects are described well by the Zipf-Mandelbrot law with an MMRE of 0.18. On the level of programming paradigms, functional languages were best described by the generalized Zipf's law while markup languages were worst.

Another conjecture that we expected to observe in the data is that the Zipf-Mandelbrot law can be used to model external documentation and bug reports well, given that these artifacts are mainly comprised of natural language. While the MMRE values for bug reports (i.e., 0.23) and documentation (i.e., 0.21) do fall into the criteria for *good fit*, it should be noted that most of the projects in miscellaneous PLs under this study resulted in lower (i.e., *better*) MMRE values. This is a result that we did not expect and answering the *why* question remains our future work.

3.2 RQ2: Validating software science equation

Since our observations lead us to conclude that the Zipf-Mandelbrot law is a good model for describing token frequency distributions in different PLs we can test if the revised software science equation applies in case of estimating length of the programs written in other PLs as well. We present the descriptive statistics on MMRE values in the Table 2.

Based on the results we conclude that the revised software science equation *does not hold* for projects across different PLs. The results of our study refute the results of the previous studies showing that the revised equation can be effectively used for modeling "vocabulary-length" relationship in Java programs. Nevertheless, our findings indicate that the best estimations were made for Smalltalk and C programs with MMREs of 0.16 and 0.17 respectively.

Unexpectedly, the Java projects used in our study (notice that we used a different set of Java projects to the one used in [18]) resulted in an MMRE of 2.23, which landed Java as the least suitable language to be estimated by the equation. These findings provide substantial evidence that additional research on the relationship between software vocabulary and program length is required before it can be formalized.

[1] http://www.cs.wm.edu/~dpierret/zipf-appendix.html

4. Conclusions and Future Work

In this paper we present the first comprehensive study of software lexicons in systems implemented in different PLs and paradigms as well as other software artifacts. Using MMRE values to measure a goodness of fit, we conclude that the Zipf-Mandelbrot law is successful in describing token distributions for all languages and paradigms, to varying degrees (**RQ1**). The law is also applicable for describing token frequency distributions in external documentation and bug reports (**RQ1**). We also found that the software science program length estimation equation [18] does not hold for projects written in different PLs (**RQ2**). Our results indicate that further studies are necessary to derive reliable software science length equation for predicting software size based on its vocabulary.

In our future work we are planning on expanding on the number of projects per language to make the findings statistically significant. We would also like to include more PLs per paradigm for similar reasons. In addition, we are planning on analyzing certain types of tokens and their positions on Zipf curves in different projects. Such data might include stop words, API calls, keywords and identifiers. Lastly, the constants used for fitting to the Zipf's law could possibly reflect certain characteristics of the PL or program domains [6]. We are interested in investigating and generalizing these characteristics in projects from diverse domains and implemented in a range of PLs.

5. Acknowledgements

We would like to thank Panagiotis Foteinos for providing C++ programs used in this study and Zheng "Eddy" Zhang for helping with the statistics gathering process. This research was supported in part by the United States Air Force Office of Scientific Research under grant number FA9550-07-1-0030.

6. References

[1] Anquetil, N. and Lethbridge, T., "Assessing the Relevance of Identifier Names in a Legacy Software System", in Proc. of Annual IBM Centers for Advanced Studies Conference (CASCON'98), Toronto, Ontario, Canada, December 1998, pp. 213-222.

[2] Antoniol, G., Gueheneuc, Y.-G., Merlo, E., and Tonella, P., "Mining the Lexicon Used by Programmers during Software Evolution", in Proc. of 23rd IEEE International Conference on Software Maintenance (ICSM'07), Paris, France, 2007, pp. 14-23.

[3] Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Visser, M., Melton, H., and Tempero, E., "Understanding the Shape of Java Software", in Proc. of Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'06), October 2006, pp. 397-412.

[4] Concas, G., Marchesi, M., Pinna, S., and Serra, N., "Power-Laws in a Large Object-Oriented Software System", *IEEE Transactions on Software Engineering (TSE)*, vol. 33, no. 10, October 2007, pp. 687-708.

[5] Deissenboeck, F. and Pizka , M., "Concise and Consistent Naming", *Software Quality Journal*, vol. 14, no. 3, 2006, pp. 261-282

[6] Gelbukh, A. F. and Sidorov, G., "Zipf and Heaps Laws' Coefficients Depend on Language", in Proc. of 2nd International Conference on Computational Linguistics and Intelligent Text Processing, 2004, pp. 332 - 335.

[7] Haiduc, S. and Marcus, A., "On the Use of Domain Terms in Source Code", in Proc. of 16th IEEE International Conference on Program Comprehension (ICPC'08), Amsterdam, The Netherlands, June 10-13 2008, pp. 113-122.

[8] Halstead, M. H., *Elements of Software Science*, Elsevier Science Inc, 1977.

[9] Hindle, A., Godfrey, M. W., and Holt, R. C., "Reading Beside the Lines: Indentation as a Proxy for Complexity Metric", in Proc. of 16th IEEE International Conference on Program Comprehension (ICPC'08), 2008, pp. 133-142.

[10] Knuth, D. E., *Sorting and searching*, Addison-Wesley, 1998.

[11] Lawrie, D., Feild, H., and Binkley, D., "Leveraged Quality Assessment Using Information Retrieval Techniques", in Proc. of 14th IEEE International Conference on Program Comprehension (ICPC'06), Athens, Greece, June 14-16 2006, pp. 149-158.

[12] Lawrie, D., Feild, H., and Binkley, D., "An Empirical Study of Rules for Well-Formed Identifiers", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, 2007, pp. 205-229.

[13] Lawrie, D., Feild, H., and Binkley, D., "Quantifying Identifier Quality: An Analysis of Trends", *Empirical Software Engineering*, vol. 12, no. 4, 2007, pp. 359-388.

[14] Louridas, P., Spinellis, D., and Vlachos, V., "Power laws in software", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 18, no. 1, 2008.

[15] Mandelbrot, B. B., "An information theory of the statistical structure of language", in Proc. of Symposium on Applications of Communications Theory, London, 1953.

[16] Pareto, V., *Cours d' Economie Politique*, Rouge, Lausanne, 1897.

[17] Yule, G. U., "A Mathematical Theory of Evolution, Based on the Conclusions of Dr. J. C. Willis, F.R.S", *Royal Society of London Philosophical Transactions* vol. 213, no. B, 1925, pp. 21-87.

[18] Zhang, H., "Exploring Regularity in Source Code: Software Science and Zipf's Law", in Proc. of 15th IEEE Working Conference on Reverse Engineering (WCRE'08), Antwerp, Belgium, 2008.