Language Entropy: A Metric for Characterization of Author Programming Language Distribution

Jonathan L. Krein, Alexander C. MacLean
SEQuOIA Lab, Brigham Young University
jonathankrein@byu.net, amaclean@byu.net

Daniel P. Delorey
Google, Inc.
delorey@google.com

Charles D. Knutson
SEQuOIA Lab
Brigham Young University
knutson@cs.byu.edu

Dennis L. Eggett
Dept. of Statistics
Brigham Young University
theegg@stat.byu.edu

ABSTRACT

Programmers are often required to develop in multiple languages. In an effort to study the effects of programming language fragmentation on productivity—and ultimately on a programmer's problem solving abilities—we propose a metric, *language entropy*, for characterizing the distribution of an individual's development efforts across multiple programming languages. To evaluate this metric, we present an observational study examining all project contributions (through August 2006) of a random sample of 500 SourceForge developers. Using a random coefficients model, we found a statistically significant correlation (alpha level of 0.05) between language entropy and the size of monthly project contributions (measured in lines of code added). Our results indicate that language entropy is a good candidate for characterizing author programing language distribution.

1. INTRODUCTION

The ultimate deliverable for a software project is a source code artifact that enables computers to meet real-world needs by solving a set of complex problems. The process of software development, therefore, involves both problem solving and communication of solutions to a computer. We believe that the languages by which humans communicate solutions to computers may in fact play a role in the complex processes by which they generate those solutions.

Baldo et al. define language as a "rule-based, symbolic representation system" that "allows us to not simply represent concepts, but more importantly for problem solving, facilitates our ability to manipulate those concepts and generate novel solutions" [1]. Although their study focused on the effects of *natural* language on problem solving, their concept of language is highly representative of those used in programming activities. Other research in the area of linguistics examines the differences between mono-, bi-, and multilingual speakers. One particular study, focusing on the differences between mono- and bilingual children, found some specific differences in the subjects' abilities to solve problems [2].

These linguistic studies prompt us to ask questions about the effects of working concurrently in multiple programming languages on the problem-solving abilities of developers. However, to begin studying that relationship, we first develop a metric that characterizes the distribution of an author's development efforts across multiple programming languages.

In this paper, we present *language entropy* as a candidate metric for measuring the distribution of languages within an author's programming contributions. We further provide preliminary support for a relationship between this metric and the problem-solving abilities of developers by demonstrating a significant relationship between language entropy and programmer productivity, as measured in lines of source code added to projects per month.

2. QUESTION OF INTEREST

What effect does working in multiple programming languages concurrently have on a programmer's productivity?

- Positive Correlation: A programmer contributing in multiple programming languages may be more productive due to his or her ability to draw from multiple programming paradigms. For example, software developers writing in a functional language such as Lisp arguably approach a problem differently than those writing in a purely object-oriented language such as Java.
- Negative Correlation: A developer contributing in more than one language may be less productive because he or she has to context switch between multiple languages.
- No Correlation: A developer's productivity may be independent of his or her programming language distribution.

3. LANGUAGE ENTROPY

In order to empirically evaluate the correlation between language fragmentation and programmer productivity, we require a metric that accurately characterizes the distribution

of an author's development efforts across multiple programming languages. In this section we present language entropy as a candidate metric, detail its calculation, and explain its behavior in response to changes in the number of languages a developer uses. For a deeper treatment of entropy as it relates to software engineering, see [8].

3.1 Definition

Entropy is a measure of chaos in a system. The concept of entropy originated in thermodynamics but has been adopted by information theory [7]. For our purposes, we use entropy as a measure of the evenness with which an author contributes in different programming languages. For example, if an author is working in two languages and splits his or her contribution evenly between the two, entropy is 1. However, a 75-25 split across the two languages yields an entropy of approximately 0.8 (see Table 1).

% Contribution		
A	**B**	**Entropy**
0	100	0
25	75	~ 0.8
50	50	1
75	25	~ 0.8
100	0	0

Table 1: Entropy Example for two languages, A and B.

3.2 Calculation

The general form for entropy is shown in Equation 1.

$$E(S) \equiv -\sum_{i=1}^{c}(p_i \cdot log_2 p_i) \quad (1)$$

In this equation, the variables are defined as follows:

- S: the system
- c: the language count
- p: the proportion of the contribution of language i to the total contributions S

The general form of entropy can be applied to any number of languages to generate an entropy value. Two languages, as shown in Figure 1, produce a parabolic curve. Three languages produce a three-dimensional shape. Entropy calculations beyond three dimensions are difficult to visualize.

To compute an author's language entropy we calculate the proportion for each programming language represented in the author's total contribution—p_i values in Equation 1[1]. We then calculate the result of Equation 1 using those language proportions.

3.3 Behavior

Language entropy characterizes the developer's fragmentation across multiple programming languages. However, because entropy is based on logarithms, its response to changes in the number of languages a developer uses is non-linear. The equation for the maximum possible entropy value for a given number of languages is shown in Equation 2, where c is the language count.

$$E_{max} = log_2(c) \quad (2)$$

Notice that the equation's maximum value increases as c increases. Thus, for each additional language in the entropy calculation, an author's maximum possible entropy value rises.[2] However, the effect of adding an additional language diminishes as the total number of languages increases (see Equation 3 and Table 2).

$$\lim_{c \to \infty} E_{max}(c+1) - E_{max}(c) = 0 \quad (3)$$

Conversely, the minimum possible language entropy is always 0, indicating that the author only added lines of code in a single language (see Table 2).

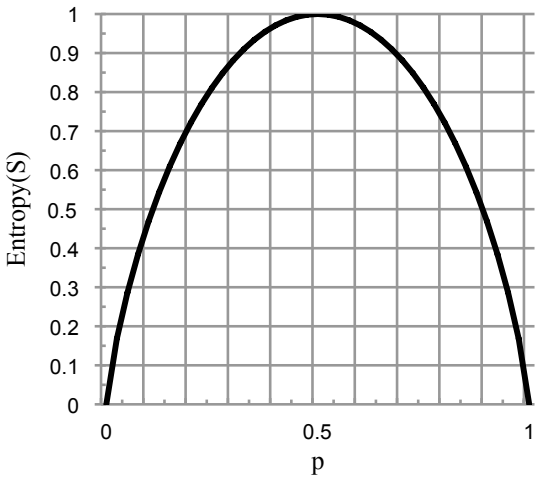

Figure 1: 2nd Order Entropy Curve

# of Languages	Min. Entropy	Max. Entropy
1	0	0
2	0	1
3	0	1.585
4	0	2
5	0	2.322
⋮	⋮	⋮
49	0	5.615
50	0	5.644

Table 2: Sample Entropy Values

4. DATA

The data set used in this study was previously collected for a separate, but related work. It was originally extracted from the SourceForge Research Archive (SFRA), August 2006. For a detailed discussion of the data source, collection tools and processes, and summary statistics, see [6].

4.1 Description of the Data Set

The data set is composed of *all* SourceForge projects that match the following four criteria: 1) the project is open source; 2) the project utilized CVS for revision control; 3) the project was under active development as of August 2006;

[1]For the purposes of this paper, contribution is defined as the number of lines of code produced per month.

[2]Note that the log operation is undefined at zero; thus, languages with a $p_i = 0$ must be excluded from the calculation.

	Project Rank	Author Rank	File Rank	Revision Rank	LOC Rank	Final Rank
C	1	1	2	2	1	1
Java	2	2	1	1	2	2
C++	4	3	4	4	3	3
PHP	5	4	3	3	4	4
Python	7	7	5	5	5	5
Perl	3	5	9	9	6	6
JavaScript	6	6	6	8	10	7
C#	9	9	7	6	7	8
Pascal	8	10	8	7	8	9
Tcl	11	8	10	10	9	10

Table 3: Top ten programming languages by popularity rankings

4) the project was in a Production/Stable or Maintenance stage. The data set includes nearly 10,000 projects with contributions from more than 23,000 authors who collectively made in excess of 26,000,000 revisions to roughly 7,250,000 files [6].

A study by Delorey, Knutson, and Chun [4] identified more than 19,000 different file extensions in the data set, representing 107 unique programming languages. The study also noted that 10 of those 107 languages are used in 89% of the projects, by 92% of the authors, and account for 98% of the files, 98% of the revisions, and 99% of the lines of code in the data set. Table 3 shows the 10 most popular languages with rankings. Delorey et al. ranked the languages based on the following 5 factors: 1) total number of projects using the language; 2) total number of authors writing in the language; 3) total number of files written in the language; 4) total number of revisions to files written in the language; and 5) total number of lines written in the language.

4.2 Producing a Data Sample

From the initial data set we extracted a random sample of 500 developers[3] along with descriptive details of all revisions that those developers made since the inception of the projects on which they worked. We then condensed this sample by totaling the lines of code added by each developer for each month in which that developer made at least one code submission. The final step in generating the sample was calculating the language entropy in each month for each developer. Note that months in which developers made no contributions are discarded due to the fact that the language entropy metric is undefined for zero lines of code.

Ignoring a developer's "inactive" months is reasonable since for this study we are more interested in whether lines of code production is related to language entropy than we are in the actual magnitude of that relationship. However, our model does assume that the code was written in the month in which it was committed. Therefore, months without submissions represent a confounding factor in this study.

To help account for multi-month code submissions, as well as the factors identified in [4], we applied several filters to the data sample. However, analyses of the filtered and unfiltered data produced approximately equivalent results. Therefore, we report our results from the more robust, unfiltered data sample.

[3]For the purposes of this study, a *developer* is an individual who contributed at least one line of code in at least one revision.

To filter the data, we 1) removed all data points of developers who submitted more than 5,000 lines of code during at least three separate months, and 2) removed all remaining data points for which the month's submission was greater than 5,000 lines of code. The first filter was intended to remove project gatekeepers, who submitted code on behalf of other developers. If a developer was suspected of being a gatekeeper, all of his/her contributions were excluded. The second filter was designed to remove significant quantities of auto-generated code.

We feel that these two filters are sufficient on the grounds that in [4], Delorey et al. ultimately controlled for outliers by capping the annual author contribution at 80,000 lines of code. Our limit of 5,000 lines of code per month results in a maximum possible annual contribution of 60,000 lines of code per author—a bit more conservative.

5. ANALYSIS

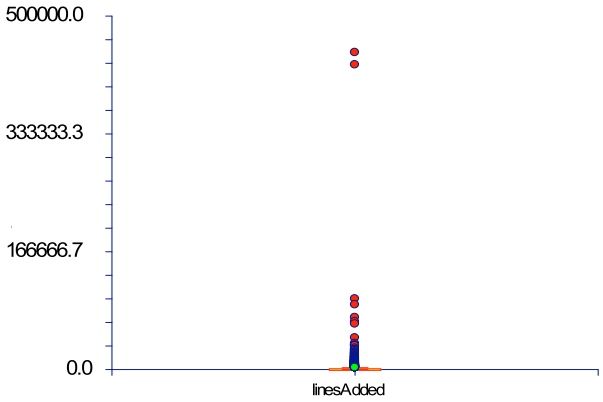

Figure 2: Box Plot of Lines Added

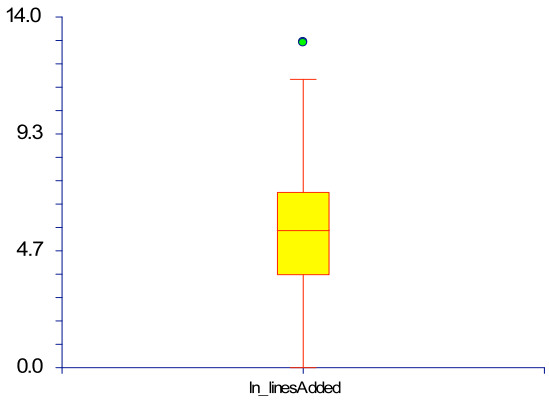

Figure 3: Box Plot of ln(Lines Added)

5.1 Transforming the Data

Figure 2 shows a box plot of the lines added. Three threats to statistical model assumptions are clearly visible: significant outliers, a skewed distribution, and a large data range. We adjust for all three issues by applying a natural log transformation. Notice in figure 3, which depicts the transformed data, that there are only minimal outliers, the range is controlled, and the distribution is approximately normal.[4]

[4]Statistical models assume specific characteristics about data. Sometimes data must be transformed before it can be

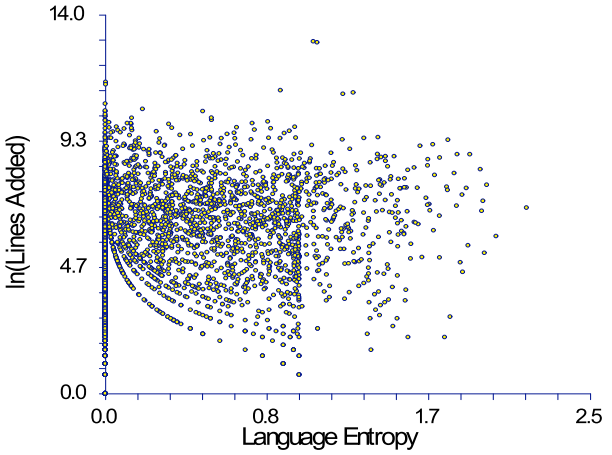

Figure 4: Plot of ln(Lines Added) vs. Language Entropy

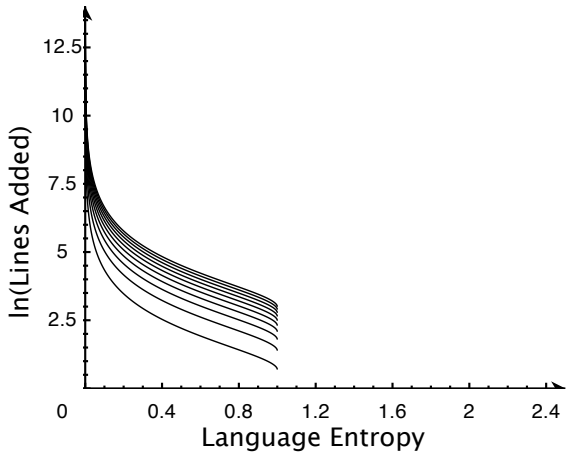

Figure 5: Graph of the First 10 Entropy Curves for the 2 Language Case

5.2 Selecting a Statistical Model

Figure 4 displays a plot of lines added (on the natural log scale) versus language entropy, in which each point on the graph represents one month of work for one developer. First, be aware that the volume and distribution of data points (see table 4) is masked by crowding, which causes points to be plotted over other points. In total, there are 3,940 points plotted, of which 1,945 points lie on the y-axis at the entropy value of zero. Thus, nearly half the data consists of months in which developers submitted code in only one language.[5]

Further, there is a pattern of curving lines visible at the bottom of the point cloud between zero and one entropy. The banding pattern is due to both the nature of the language entropy calculation and the lines added. Specifically, the two metrics partition the data points into equivalence classes, one for each band on the graph. Data points in the first equivalence class—the band closest to the x-axes—correspond to monthly contributions in which all lines but one were written in the same language. Data points in the second equivalence class correspond to monthly contributions in which all but two lines were written in one language. By the fourth equivalence class the bands are so close that they blend together on the graph. Figure 5 shows a graph of the first 10 equivalence classes.

accurately analyzed. However, interpretation of the results must reflect the transformation. In this study, for instance, the slope of a regression line must be interpreted as a multiplicative factor since the dependent variable is logged.

[5]The distribution of the data points with respect to language entropy is fairly consistent with [5], in which the authors (referring to the data set of this study) note that for approximately 65% of projects developers submit code in a single language per year. For 20% and 12% of projects, developers submit code in two and three languages per year respectively.

LE Range	Data Points
$LE = 0$	1,945
$0 < LE \leq 1$	1,705
$1 < LE$	290
Total Data Points:	3,940

Table 4: Distribution of Data Points by Language Entropy (*LE*)

The scatter plot also exhibits a vertical boundary of points just before entropy of one. This pattern is possibly due to the sparseness of the data beyond entropy of one (only 290 points).

It is immediately apparent that the distribution of the data is different during months in which developers contributed code in only one language (zero entropy), versus months in which they contributed code in more than one language (greater than zero entropy). Therefore, it would be inappropriate to apply a simple linear regression model to the full range of the data. Instead, we use a *random coefficients model* which allows us to estimate a mean for the group at zero entropy, as well as fit a regression line to the rest of the data. These two groups could be analyzed separately, but fitting them under one model allows us to pool the data when computing the error terms, which results in tighter confidence intervals and a more efficient analysis.

5.3 Adjusting for Serial Correlation

A final concern is the potential for serial correlation in the data (i.e., the data correlates with itself) as a result of the measurements being taken over time. Estimating the mean of data that is self-correlated requires statistical adjustment in order to produce accurate results. The data sample in this study contains an average of eight months of measurements per developer, which is insufficient to confidently identify a serial correlation. However, to be conservative we assume serial correlation is present in the data and account for it in our analysis.

6. RESULTS

Table 5 shows estimates (on the natural log scale) of the model parameters, with confidence intervals and two-sided p-values. All three parameters are statistically significant with p-values less than 0.0001. Such small p-values allow us to confidently conclude that the relationship between language entropy and lines added is *not* due to random chance. The low error terms, which result in narrow confidence intervals around the parameter estimates, give us confidence that our sample size is sufficient to accurately estimate the

Parameter	Estimate	Lower 95% CL	Upper 95% CL	p-value	Standard Error	DF
zeroEntropyGroupMean	4.0690	3.9208	4.2172	<.0001	0.07542	425
nonZeroEntropyGroup	2.2870	2.1014	2.4726	<.0001	0.09411	196
nonZeroEntropySlope	-0.6963	-0.9646	-0.4280	<.0001	0.13600	181

Table 5: Model Parameter Estimates

population variance. Further, since the data sample was randomly selected (as described in section 4.1), we conclude that the patterns in the data sample characterize the entire SourceForge population. However, since this is an observational study, we cannot infer causality. Therefore, the remainder of the discussion of results describes the observed relationship between language entropy and lines added.

In Table 5, the *zeroEntropyGroupMean* is an estimate of the mean of the data points at zero language entropy (the zero group, or ZG). The *nonZeroEntropyGroup* represents the estimated difference between the ZG mean and the intercept of the regression line for the non-zero entropy data (the non-zero group, or NZG). The very low p-value for this parameter indicates that the ZG mean is significantly different from the trend in the NZG. Adding the first two parameter estimates gives the estimate for the intercept of the NZG regression line (6.3560). The third parameter, *nonZeroEntropySlope*, represents the slope of the NZG regression line, which is negatively correlated with language entropy.

The magnitudes of these parameter estimates make more sense on the original scale. However, because the analysis is performed on log-transformed data, the back-transformed estimates must be interpreted differently. Specifically, the ZG mean and the intercept of the NZG regression line both represent medians on the original scale. Also, the slope of the NZG regression line becomes a multiplicative factor, which means that an increase in language entropy results in a multiplicative increase in lines added.

Thus, for months in which a developer submits code in one language (ZG), the developer contributes, on average, 58 lines of code (95% confidence interval from 50 to 68 lines of code). However, extrapolating the trend in the NZG, which represents months in which developers submitted code in more than one language, one would expect the ZG median to be 576 lines of code—a significant difference. Note, though, that this difference considers both highly and marginally active developers equally. The marginally active developers, who make only a few small contributions, and for whom a productivity increase is less interesting, may be significantly pulling down the ZG median (See section 7.4 for further discussion).

Lastly, for months in which a developer submits code in more than one language, the developer's monthly contributions decrease by an estimated 6.7% for each 0.1 unit increase in language entropy. For a 1.0 unit increase in language entropy, a developer's monthly contribution drops by approximately 50% on average.

7. LIMITATIONS

In the following subsections we identify several limitations of this study.

7.1 Non-Contributing Months

The developers in our data set did not always contribute to projects in contiguous months. For example, a developer might contribute changes in April, skip May, and contribute again in June. For the purposes of this study we assumed that developers submitted contributions in the same months in which those contributions were written. We took steps to help ensure our assumption (see Section 4.2). However, we do not have an empirical foundation for applying a cap of 5,000 lines to monthly programmer contributions. Also, we have not empirically validated our method of identifying gatekeepers.

7.2 SourceForge

Our inferences are limited to developers on SourceForge. Therefore, we cannot make general conclusions about other software development environments. Also, the SourceForge archive obscures certain information about developers (such as the identity of gatekeepers).

7.3 Productivity Measure

Despite its utility in this preliminary study, lines of code is a weak measure of programmer productivity. Further studies should extend the analysis of language entropy to other productivity models.

7.4 Marginally Active Developers

Developers who make only small contributions per month may bias the analysis results. Such developers are probably less likely to write in multiple languages in a given month, in which case filtering marginally active developers could reduce the disparity between the estimated mean of the group who wrote in only one language and the trend of the remaining data. Thus, it would be interesting to add an indicator variable to the model to distinguish such developers from those who regularly contribute more significant volumes of code.

8. FUTURE WORK

In this section we outline avenues for future research.

8.1 Establishing Causality

This study establishes a correlation between language entropy and the size of developer contributions for the SourceForge population. To understand the cause of the observed relationship we need to run controlled, randomized experiments. We believe that such efforts, in combination with corporate case studies (as described in section 8.2), will provide meaningful results from which practitioners may make better-informed decisions regarding project-developer assignments and the adoption of new languages and frameworks.

8.2 Corporate Case Studies

Running a more robust analysis of language entropy utilizing data from industry projects would allow us to expand our inferences into the corporate domain, at which point we could ask a number of important questions, including:

- If my company is already maintaining a large code base in COBOL, how would my developers' productivity be affected by an additional project in Java?
- My company already supports products in different languages. Will my developers be more productive if I assign each one to a specific language, as opposed to spreading them across languages?

8.3 Paradigm Relationships

Many of the languages in our study cluster by paradigm (Java, C++, and C#, for example). Switching between programming languages that share a common paradigm may not be as cognitively difficult as switching between languages from different paradigms. We expect changes in entropy to affect a programmer working within a single paradigm less than one working across multiple paradigms.

8.4 Commonly Grouped Languages

In this study we examine the effect of language entropy on productivity across all languages. However, some languages are commonly used together (e.g., many web projects are based on Java, JavaScript, and HTML). Is the cognitive burden of context switching between languages reduced for developers who work across a set of commonly grouped languages?

8.5 Language Entropy as a Productivity Measure

To better understand the relationship between language entropy and other productivity metrics, we need to determine whether language entropy provides new information beyond the metrics already presented in the literature. If shown to be complementary, language entropy can be incorporated into more complex productivity models [3].

9. CONCLUSIONS

The results of this study suggest a correlation between language entropy and programmer productivity. However, because our study is observational, we cannot infer that the differences in language entropy caused the observed variation in productivity. Nevertheless, since the data was randomly selected, we can make inferences to the general SourceForge community for those developers who actively worked on Production/Stable or Maintenance projects from 1995 through August 2006. Specifically, we can make two inferences:

1. For those developers who wrote in multiple languages, higher language entropy is negatively correlated with the number of lines of code contributed per month.
2. For months in which developers submitted code in a single language, their contributions were significantly smaller than the trend suggested by the rest of the data.

The primary objective of this study was to develop a metric with which we could investigate the relationship between an author's ability to solve software problems and the distribution of programming languages within his or her project contributions. The relationship between language entropy and productivity in this initial study demonstrates that language entropy is a good candidate for measuring the distribution of an author's development efforts across multiple programming languages. This result, therefore, justifies further research into the relationship between language entropy and the problem-solving abilities of developers.

10. REFERENCES

[1] Juliana V. Baldo, Nina F. Dronkers, David Wilkins, Carl Ludy, Patricia Raskin, and Jiye Kim. Is problem solving dependent on language? *Brain and Language*, 92(3):240–250, 2005.

[2] Ellen Bialystok and Shilpi Majumder. The relationship between bilingualism and the development of cognitive processes in problem solving. *Applied Psycholinguistics*, 19(1):69–85, 1998.

[3] L.C. Briand, S. Morasca, and V.R. Basili. Defining and validating measures for object-based high-level design. *Software Engineering, IEEE Transactions on*, 25(5):722–743, Sep/Oct 1999.

[4] Daniel P. Delorey, Charles D. Knutson, and Scott Chun. Do programming languages affect productivity? a case study using data from open source projects. In *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS '07)*, May 2007.

[5] Daniel P. Delorey, Charles D. Knutson, and Christophe Giraud-Carrier. Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects. In *Second International Workshop on Public Data about Software Development (WoPDaSD '07)*, June 2007.

[6] Daniel P. Delorey, Charles D. Knutson, and Alex MacLean. Studying production phase sourceforge projects: A case study using cvs2mysql and sfra+. In *Second International Workshop on Public Data about Software Development (WoPDaSD '07)*, June 2007.

[7] W. Harrison. An entropy-based measure of software complexity. *Software Engineering, IEEE Transactions on*, 18(11):1025–1029, Nov 1992.

[8] Quinn C. Taylor, James E. Stevenson, Daniel P. Delorey, and Charles D. Knutson. Author entropy: A metric for characterization of software authorship patterns. In *Third International Workshop on Public Data about Software Development (WoPDaSD '08)*, page 6, September 2008.