Author Entropy: A Metric for Characterization of Software Authorship Patterns

Quinn C. Taylor, James E. Stevenson, Daniel P. Delorey, and Charles D. Knutson
SEQuOIA Lab, Brigham Young University
2236 TMCB
Provo, Utah 84602
{qtaylor, jstevenson}@byu.net, {pierce, knutson}@cs.byu.edu

ABSTRACT

We propose the concept of *author entropy* and describe how file-level entropy measures may be used to understand and characterize authorship patterns within individual files, as well as across an entire project. As a proof of concept, we compute author entropy for 28,955 files from 33 open-source projects. We explore patterns of author entropy, identify techniques for visualizing author entropy, and propose avenues for further study.

1. INTRODUCTION

Software development is a process fraught with complexity and unpredictability because software is designed and written by people. Human interactions add complexity to development processes, although some software engineering authorities disagree about the implications [6, 7, 8].

Contributor interactions critically affect software development, and it follows that characterizing contributor interaction is an important task. Studies of developer interactions have generally focused on bug tracking, mailing list analysis, or studies of developer productivity; few consider developer interaction within source code.

In this paper, we introduce *author entropy*, a metric that quantifies the mixture of author contributions to a file. Just as code-level metrics—including file length, number of function points, complexity, cohesion, and coupling—quantify properties of source code, author entropy characterizes properties of author interactions within source files using a simple summary statistic.

This paper describes the author entropy metric, presents a proof of concept empirical study, and proposes topics for future research relating to author entropy and authorship patterns.

2. AUTHOR ENTROPY

In this section, we discuss entropy, define how entropy is calculated, and describe how entropy applies to authorship.

2.1 Definitions of Entropy

Entropy is a measure of chaos or disorder in a system. Thermodynamics defines entropy as a measure of randomness of molecules in a system; an increase in entropy leads to greater spontaneity. Entropy is often understood as the "useless energy" in a system: energy which is not available to perform work. The second law of thermodynamics states that the entropy of an isolated system will tend to increase over time, approaching a maximum value at equilibrium.

Information theory borrows the idea of entropy, defines it in terms of probability theory, and uses it to analyze communication, compression, information content, and uncertainty [16]. In machine learning, entropy "characterizes the (im)purity of an arbitrary collection of examples", or the degree to which members of a collection can be split into groups based on a given attribute [14].

Entropy may also be applied to software engineering as a measure of collaboration. Specifically, we consider entropy of source code, which can be broken into smaller segments (e.g., lines, functions, statements, identifiers, etc.) and classified by author. This definition of entropy allows us to quantify the mixture of author contributions to a file. We discuss why this matters to software in Section 2.3.

2.2 Calculating Entropy

Entropy is a summary statistic[1] calculated from the relative sizes of the groups or classifications present in a system. Entropy formulae are nearly identical across domains, varying only in constant multipliers and symbolic representation.

2.2.1 The Special Case: Binary Classification

We first consider entropy in the special case of a Bernoulli distribution with proportion p of positive outcomes and proportion q of negative outcomes, where $0 \leq p \leq 1$. The entropy of system S (shown in Figure 1) is defined as:

$$E(S) \equiv -p \cdot log_2\ p\ - q \cdot log_2\ q \tag{1}$$

Note that entropy is maximized when p and q are equal, and minimized when either proportion reaches 1.

2.2.2 The General Case: Any Number of Groups

Entropy also generalizes to an arbitrary number of groups. If elements of a system S belong to c different classes, and p_i is the proportion of elements in S belonging to class i, then the entropy of S is:

$$E(S) \equiv -\sum_{i=1}^{c}(p_i \cdot log_2\, p_i) \tag{2}$$

[1]Summary statistics are lossy summaries of observations, such as mean, median, variance, skewness, and kurtosis.

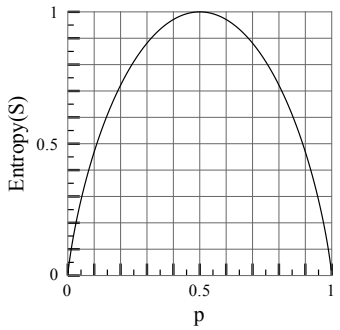


Figure 1: Entropy of a Bernoulli distribution.

$E(S)$ is maximized when the proportions of classes in S are equal ($\forall\ i, p_i = \frac{1}{c}$). Equation 2 is a non-normalized summation, so the limit of $E(S)$ is a function of c. As shown in Figure 2, if the elements of a system S belong to c possible classes, the entropy can be as large as:

$$E_{max}(S) \equiv log_2\, c \tag{3}$$

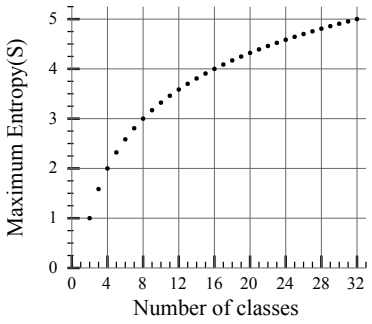


Figure 2: Maximum possible entropy for a system S as a discrete function of the total number of groups.

Because the maximum possible entropy for a system is a function of c, intuitive understanding of an entropy value can be difficult. For example, an entropy of 1 is the maximum entropy for a system with 2 classes, but comparatively low entropy for a system with 10 classes. Dividing $E(S)$ by $log_2\, c$ produces a value in the range [0,1]; this normalized entropy value represents the percentage of maximum entropy, which may be more intuitive than non-normalized entropy. (See section Section 5.3 for normalization strategies.)

By definition, a system with only one class has zero entropy, so we define $log\, 0$ to be 0. We also note that the logarithmic base is unrelated to the number of classifications; we use log_2 for historical reasons rooted in information theory.

2.2.3 Entropy Applied to Text Authorship

If system S is a file and c is the number of authors, then each p_i is the proportion of the text written by author i, and $E(S)$ is the entropy of the file. The values of p_i are the proportions of text segments attributed to each author. (Text segments may be of any size, although entropy may be more meaningful when the segments are roughly equivalent in size or information content.) Entropy increases as all authors' contributions (p_i) approach equality.

2.3 Interpretation of Entropy in Software

Entropy in source code is not inherently good or bad; it merely indicates that multiple people are contributing in a fairly balanced way. Although low entropy could be an indicator of modular team structure and well-architected software, it could also reflect poorly structured code that few contributors are willing to work on. Similarly, high entropy could be the result of poor communication or code that is in dire need of refactoring, or it may indicate excellent organization that makes it easy for many authors to contribute to the same code. As with any metric, context is essential.

Correlating entropy with other metrics and observations can provide valuable new insights. For example, a file with high entropy written by several experts may be of higher quality than a file written by one novice author; combining entropy with a metric of quality can help distinguish between "good entropy" and "bad entropy". Several ways to leverage the author entropy metric are discussed more in Section 5.1.

Author entropy cannot directly indicate attributes of the subject text. For example, file length is obscured since files of different size but equal proportions of contribution have the same entropy. Entropy also does not consider quality or the relative importance of contributions, such as new functionality, bug fixes, comments, whitespace, or formatting.

3. PROOF OF CONCEPT STUDY

In order to give the reader a better understanding of how author entropy can be useful, we have conducted a small empirical study as a proof of concept that demonstrates possible applications of the metric. We begin by describing the methods and tools used to gather data and calculate author entropy. As part of that discussion we present the criteria we used to select projects for the study and the threats to the validity of our results. We also present some results of our preliminary study, including observations and analysis of authorship patterns manifest in the data.

3.1 Extraction and Calculation

Author entropy calculations require data that attributes text fragments to authors. Software authorship information can be gleaned from revision control systems that record snapshots of development history [4]. Our exploratory study considers only projects stored in Subversion.

We created a Python script to collect author data. We use Subversion's `log` command to identify the files modified in each revision, and record the revision number and path for each file. We then use Subversion's `blame` command to determine authorship for each line in each changed file. Author counts are divided by the total number of lines in the file to obtain p_i values, and author entropy for each file is calculated as shown in Equation 2. Entropy for each file is also normalized to the range [0,1] as in Equation 3.

3.2 Project Selection

Due to the amount of data which must be analyzed, we identified a subset of SourceForge projects with favorable characteristics. (We selected SourceForge.net because it hosts thousands of projects with multiple years of development history and various development platforms.) Howison [12] has identified potential weaknesses in this approach.

Many projects were not suitable for our analysis because they 1) were immature, abandoned, or not very active, 2)

didn't use Subversion exclusively, 3) had very few developers, or 4) contained many non-source text files. We addressed these issues by limiting our sample to projects that meet the following criteria:

1. Projects categorized as "Production/Stable".
2. Projects registered since 2006 (higher SVN usage).
3. Projects with 5 or more committing developers.[2]
4. Projects in Java with easily identifiable source files.

We queried FLOSSmole [1] data with these four criteria and identified 33 candidate projects, with the following distributions of revisions and authors:

	Min	Q1	Median	Q3	Max
Revisions	41	373	723	994	11576
Authors	5	6	8	13	23

Table 1: Distributions of revisions and authors for 33 projects selected from SourceForge.

3.3 Threats to Validity

One significant concern is the limited number of projects and the criteria used to select them. Although many open-source projects share similar development patterns, by no means should our results be construed as representative of all open-source projects, or even of all projects hosted on SourceForge. Many projects that did not fit our criteria would undoubtedly exhibit interesting authorship patterns.

Hidden factors which we have not addressed include irregularities in the historical data. For example, some projects contained anonymous commits, and many had a majority of commits from a single author. Without more in-depth study of specific projects, we cannot ascertain whether one developer in fact wrote all the code, or whether other contributors submitted patches that a single developer then committed. We also did not examine any specific changes to see whether changes in entropy were caused by source code reformatting, which artificially attributes lines to the committing author.

Our study is limited to line-level granularity provided by Subversion, and does not examine how much of a line changes.

With these threats to validity, however, it is important to reiterate that the focus of this paper is the author entropy metric itself. Our study is intended as a proof of concept, and should not be interpreted as exhaustive or complete. We describe potential avenues for future research in Section 5.

3.4 Results

In this section, we identify and offer possible explanations for patterns we observed in our study. These observations place author entropy in a real-world context; we demonstrate how changes to individual files affect entropy, characterize relationships between number of authors and entropy distribution, and identify project-wide entropy patterns.

[2]We scraped Subversion logs to determine the actual number of committing authors, instead of relying on the number of registered developers.

3.4.1 Degree of Collaboration Within Files

For the projects in our sample, the maximum number of authors contributing to a project was 23, but there were no individual files with more than 9 authors. Figure 3 shows the counts of file revisions and unique files we observed with each number of authors, plotted on a logarithmic scale.

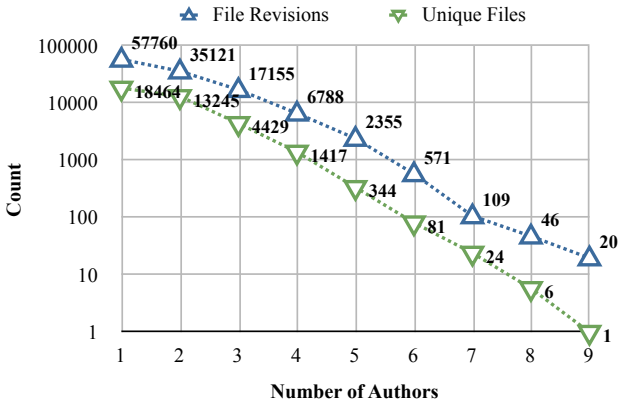


Figure 3: Counts of file revisions and unique files plotted against number of authors.

We found it noteworthy that the counts of file revisions and unique files with n authors are inversely proportional to n and exhibit near-perfect exponential decay. We hypothesize that communication and coordination become prohibitively expensive as the number of authors increases, and that these costs naturally discourage many authors from contributing on a file.

3.4.2 Entropy Patterns Within Files

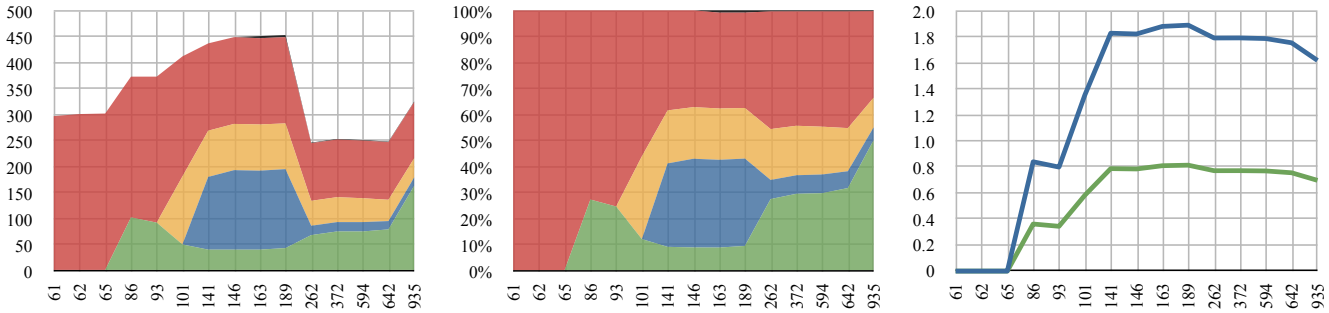
We also focused on fine-grained analysis of individual files to identify potentially interesting entropy patterns. We chose files that had high standard deviation of normalized entropy over multiple revisions (an indicator of significant changes) and compared author contributions (both the number and percent of total lines) to entropy and normalized entropy. The results for one such file are shown in Figure 4.

In Figure 4(c), the upper line is raw entropy and the lower line is normalized by $log_2 5$, since 5 is the maximum number of authors that ever contributed to the file. Normalized entropy plateaus at approximately 0.8 before decreasing slowly. Note that, despite a significant reduction in number of total lines at revision 262, author entropy does not drop rapidly. However, the addition of new authors at revisions 86, 101, and 141 does cause a significant increase in entropy.

Because entropy calculations include logarithmic factors, entropy is very sensitive to small segments of text added by additional authors, but less sensitive to changes once an author is "established." Consider the two author case in Figure 1: 50% of maximum entropy is reached when one author contributes approximately 10% of the text. This bias makes entropy highly sensitive to initial changes by new authors.

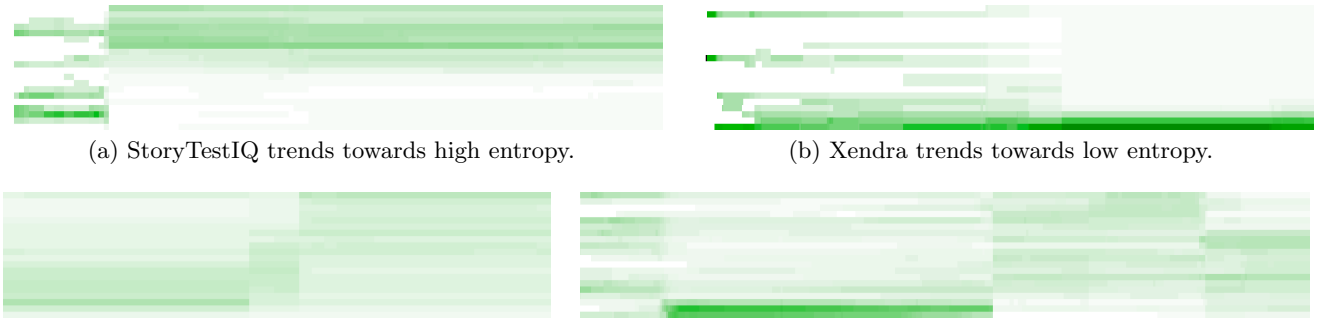
3.4.3 Entropy Distributions Within Projects

Entropy is difficult to visualize for projects with many file revisions, so we created a histogram-based plot to display entropy distributions over a project's life. We found that using color rather than a 3D height map improved scale determination and trend exploration for large projects.



(a) Number of total lines by author. (b) Percentage of total lines by author. (c) Entropy and normalized entropy.

Figure 4: Data for 15 revisions of `/trunk/sscf/WEB-INF/src/org/corrib/s3b/sscf/manage/BookmarksHelper.java` in S3B. The x axis shows consecutive revisions; actual time periods between revisions is not represented here.



(a) StoryTestIQ trends towards high entropy. (b) Xendra trends towards low entropy.

(c) NakedObjects increases in entropy. (d) SweetDEV RIA "flip-flops" between high and low entropy.

Figure 5: Excerpts of entropy distributions for several projects, excluding zero-entropy values. The darkness of each (x, y) point represents the percentage of files at revision x that map to normalized entropy y. These plots have 20 bins over the range of entropy values and have been contrast-adjusted for better readability.

Although non-zero entropy often approximated a uniform distribution as projects progressed, several projects had patterns of generally high or low entropy, dramatic changes in entropy, and even "flip-flops" between high and low entropy.

Because the plots in Figure 5 are histograms, many files need to change before the histogram changes significantly. Dramatic shifts in entropy can occur when: 1) entropy shifts in a significant number of files or 2) a large number of files are added or removed. Development activities that may cause these shifts include: new authors contributing to existing files, refactoring, code formatting, or bug fixes.

3.4.4 *Entropy Distributions Across Projects*

We examined the distribution of entropy as the number of authors for a file increases, shown in Figure 6. For $n = 2...9$ authors, we calculated univariate Gaussian kernel density estimators (a form of histogram smoothing) for normalized and non-normalized entropy values. We then combined each density function into a single 3D plot.

The entropy distribution for files with two authors was bimodal. Files were most likely to have either: 1) very low entropy, indicating that one author contributed only a very small portion of the file, or 2) very high entropy, indicating that both authors contributed almost equally. However, the entropy distributions for more than two authors were unimodal with a mean that increased with the number of authors.

Normalized entropy for three or more authors displayed an interesting trend. As the number of authors increased, the distribution of normalized entropy remained fairly constant with a peak around 0.6. Although entropy increases as more authors are added, it remains proportional to maximum possible entropy. This may indicate hidden communication or social factors that naturally keep entropy around 60% of its maximum when more than two people contribute to a file.

3.5 Summary

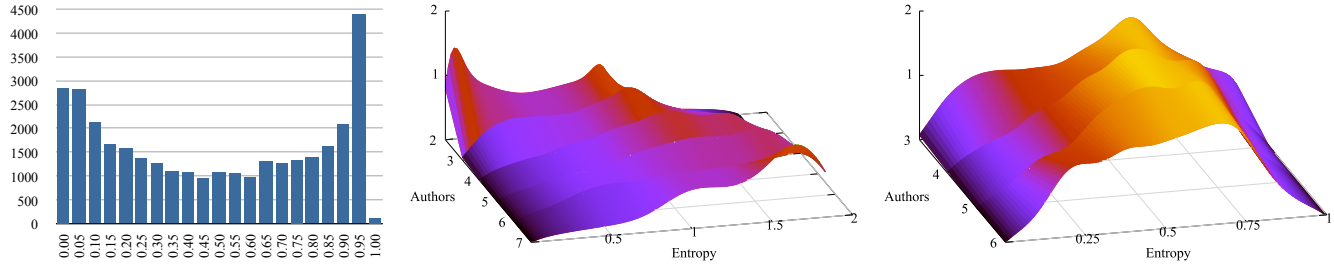
In this preliminary empirical study, we have identified several fine- and coarse-grained authorship patterns present in the projects we selected. We have observed a bimodal distribution of entropy for files with only two authors, but noted that the entropy distribution for files with three or more authors is unimodal. We have also noticed significant shifts in project-wide entropy for some projects.

While explanation of the causes of these observations is beyond the scope of this paper, these patterns (and the questions they raise) suggest that author entropy is a potentially valuable metric for software engineering researchers.

4. RELATED WORK

Several existing tools include functionality for visual or numerical analysis of authorship patterns.

CVSscan [17] is a visualization tool for observing source code structure and evolution during software maintenance.



(a) Entropy distribution, 2 authors. (b) Entropy distributions, 2–7 authors. (c) Normalized entropy, 3–6 authors.

Figure 6: Plots of entropy distributions for 28,955 files from 33 open-source projects.

It extracts data from CVS repositories and represents source code lines of one file at a time as a sequence of stacked lines that are colored according to author, age, or code construct.

StatSVN [2] retrieves information from Subversion repositories and generates charts, tables and statistics which describe project development. A few of the statistics address authorship patterns (e.g. number of source lines contributed per author, commit activity) at the directory level.

CodeSaw [10] is a tool for visualizing author activity in distributed software development. It combines code activity and developer communication to reveal group dynamics. Data for up to eight authors may be visualized together on a timeline. Hovering over the timeline displays detailed information for the selected developer and time period.

Author entropy can be coupled with the functionality in these and other tools to provide additional context for understanding established software metrics and patterns. We explore several possible options and benefits in Section 5.1.

5. FUTURE WORK

Our initial research generated many questions about the implications of author entropy in software engineering and other domains. Although we cannot fully address these topics in this paper, we list several avenues for future research.

5.1 Empirical Evaluation of Applicability

We hypothesize that high author entropy may be correlated with existing software metrics, such as lines of code, high complexity, high coupling, low cohesion, high bug count, etc. However, no work has yet been done to empirically test such theories. Studies of author entropy in the context of Lotka's Law [13] and exponential decay (similar to work done by Newby et al. [15], but in the context of files rather than across projects) may also provide significant new insights and allow characterization of developer productivity, both in terms of quantity and significance of work.

Author entropy may also lead to new insights if combined with similar metrics in other fields, such as Gini's inequality coefficient [11]. For example, balanced project contribution can mitigate "bus factor" risks, but could also be detrimental to a team's efficiency and agility if taken to extremes.

A topic of special interest to us is analysis and visualization of relationships between author entropy and program structure. Mapping entropy values to a representation of a program's structure may reveal valuable information about its evolution, similar to the findings of Gall and Lanza [9]. Author entropy could also be an effective indicator of design and behaviors which substantiate Conway's Law [7].

5.2 Aggregating Entropy for Groups of Files

In this paper we have calculated author entropy only for individual files. Entropy can also be calculated and analyzed for groups of files—such as packages, directories, modules, and projects—at any level of depth in a project hierarchy.

However, care must be taken as to how aggregated entropy is calculated. Two possible techniques we have identified are to 1) create an average or linear combination of file entropies, or 2) calculate entropy from a sum of author counts. Both techniques can pre-compute data to be used in calculations for later revisions, but the results can differ significantly between the two. Specifically, author entropy for a group of files with multiple authors should be non-zero, but the first approach does not always yield non-zero values.

For example, consider n files, each of which is written exclusively by a different author. The entropy for each individual file is zero, so any combination or average will always be zero. (In fact, when any of the files in a group have zero entropy, results from the first approach will be inaccurate.) The second approach represents overall author contributions more accurately, but does require that author counts for the most recent revision of each file in the group be stored.

Software developers are often unaware of exactly how much their programming efforts overlap with others. The ability to aggregate entropy can help more effectively evaluate and react to collaboration patterns at any level of granularity.

5.3 Normalizing Author Entropy

Comparison of files or revisions with different numbers of authors (and thus different maximum entropies) can be difficult or unintuitive. Normalization facilitates comparison between files by dividing observed entropy by maximum possible entropy, scaling entropy to the range [0,1]. However, maximum possible entropy can vary according to context, and normalization factors must be chosen carefully. For example, consider the following possible options for normalizing author entropy for a set of three files with authors {A,B}, {B,C,D}, and {E,F,G}:

1. Normalize each file's entropy by log_2 of the number of authors in that file; scales all values to the range [0,1].
2. Normalize all entropies by $log_2 3$, since the maximum number of authors in any file is 3.
3. Normalize all entropies by $log_2 7$, since there are a total of 7 unique authors between all the files.
4. Do not normalize at all; define normalized entropy as ambiguous for sets with unknown maximum entropy.

Each of these strategies has advantages and drawbacks which depend on context and the question being asked. For example, the third approach produces deceptively low entropy values when there are many unique authors and few authors per file, while the first two approaches can distort the fact that files with more authors arguably have more complex collaboration. In the first three normalization techniques, adding more files with common or unique authors can change the normalization factor.

For example, we found files with a near-even split between two authors and near-maximum entropy. The addition of a few lines from a third author raised entropy slightly, but dividing by $log_2 3$ reduced normalized entropy significantly. When the new lines were changed by one of the original authors, entropy rebounded. In such cases, examining the percent of possible entropy may detract from accurate understanding of entropy trends.

5.4 Parallels with Social Network Studies

Social network analysis is an important corollary to author entropy. It is quite likely that underling social structure influences code collaboration.

Crowston and Howison [8] have studied communication patterns in FLOSS (Free/Libre and Open Source Software) projects by examining developer interaction in bug tracking systems. They define and examine "centrality"—the degree to which communication pathways flow through a single developer. Centrality could augment author entropy data by providing social explanations for high or low entropy.

Bird et al. [5] examine hidden social structures in open-source projects. They extract latent structure from email data, show that sub-communities form within projects, and demonstrate that sub-communities are correlated with collaboration behavior. Additionally, they discuss parallels with Conway's Law [7] and Brooks' assertion that communication channels increase as the square of group size [6]. Identification of sub-communities, organizational structure, and communication channels may strengthen our hypothesis that author entropy is influenced by social structure.

Alonso et al. [3] study distinctions between open-source developers and contributors, and characterize roles of project participants based on rights to contribute. They mine CVS data for code authors and use email data to correlate coding productivity and mailing list activity, then construct interactions between contributors and committing developers. Their results could extend the author entropy metric; instead of counting only committing developers, indirect email contributors could be included in the entropy calculation.

6. SUMMARY

Author entropy is a summary statistic that characterizes contribution patterns in source code. Entropy is easy to calculate, and can be calculated for different levels of granularity (e.g., lines, methods, files, modules). While author entropy does not directly imply a level of code quality, it can be used in conjunction with other software metrics to identify potential areas of concern within the source code of a project.

In a proof of concept study, we calculated author entropy and analyzed authorship patterns for a selection of open source data. Our exploratory research revealed interesting patterns in entropy distributions which may be indicators of significant development activities.

A potentially promising area of future research is to examine author entropy in the context of social network factors such as sub-communities and communication patterns. Crowston and Howison [8] assert, "it is wrong to assume that FLOSS projects are distinguished by a particular social structure merely because they are FLOSS." The analysis of author contribution patterns in source code can help identify latent interactions and implicit social structures.

Because author entropy is a new metric, there are many unanswered questions about its utility and applicability. The vast amount of publicly available software data makes open source software research an especially suitable avenue for discovering the answers to these questions and expanding our current understanding of software development patterns.

7. REFERENCES

[1] FLOSSmole, 2004. `http://ossmole.sourceforge.net/`

[2] StatSVN, 2006. `http://statsvn.org/`

[3] O. Alonso, P. T. Devanbu, and M. Gertz. Extraction of Contributor Information from Software Repositories. 2006. Online: `http://wwwcsif.cs.ucdavis.edu/~alonsoom/`.

[4] T. Ball, J.-M. Kim, A. A. Porter, and H. P. Siy. If Your Version Control System Could Talk.... In *Workshop on Process Modelling and Empirical Studies of Software Engineering (Co-located with ICSE '97)*, May 1997.

[5] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Chapels in the Bazaar? Latent Social Structure in OSS. In *FSE 2008: 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Nov 2008.

[6] F. P. Brooks, Jr. *The Mythical Man-Month: Essays on Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Jan 1975.

[7] M. E. Conway. How Do Committees Invent? *Datamation*, 14(4):28–31, Apr 1968.

[8] K. Crowston and J. Howison. The social structure of Free and Open Source software development. *First Monday*, 10(2), Feb 2005. `http://firstmonday.org/issues/issue10_2/crowston/`.

[9] H. C. Gall and M. Lanza. Software Evolution: Analysis and Visualization. In *ICSE '06: 28th International Conference on Software Engineering*, pp. 1055–1056, May 2006.

[10] E. Gilbert and K. Karahalios. CodeSaw: A Social Visualization of Distributed Software Development. In *INTERACT 2007: 11th IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 303–316, Sep 2007.

[11] C. Gini. Variabilità e mutabilità. In *Studi Economico Giuridici della Reale Università di Cagliari*, 1912.

[12] J. Howison and K. Crowston. The Perils and Pitfalls of Mining Sourceforge. In *MSR '04: Workshop on Mining Software Repositories*, May 2004.

[13] A. J. Lotka. The frequency distribution of scientific productivity. *Journal of the Washington Academy of Sciences*, 16(12):317–324, Jun 1926.

[14] T. M. Mitchell. *Machine Learning*, pp. 55–57. McGraw-Hill, 1997.

[15] G. B. Newby, J. Greenberg, and P. Jones. Open Source Software Development and Lotka's Law: Bibliometric Patterns in Programming. *Journal of the American Society for Information Science and Technology*, 54(2):169–178, Jan 2003.

[16] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 623–656, Jul & Oct 1948.

[17] L. Voinea, A. Telea, and J. J. van Wijk. CVSscan: Visualization of Code Evolution. In *SoftVis '05: ACM Symp. on Software Visualization*, pp. 47–56, May 2005.