An Analysis of Author Contribution Patterns in Eclipse Foundation Project Source Code

Quinn C. Taylor, Jonathan L. Krein, Alexander C. Maclean, Charles D. Knutson

An Analysis of Author Contribution Patterns in Eclipse Foundation Project Source Code

Quinn C. Taylor, Jonathan L. Krein,
Alexander C. MacLean, and Charles D. Knutson

SEQuOIA Lab — Brigham Young University — Provo, Utah, USA
{quinntaylor,jonathankrein,amaclean}@byu.net, knutson@cs.byu.edu

Abstract. Collaborative development is a key tenet of open source software, but if not properly understood and managed, it can become a liability. We examine author contribution data for the newest revision of 251,633 Java source files in 592 Eclipse projects. We use this observational data to analyze collaboration patterns within files, and to explore relationships between file size, author count, and code authorship. We calculate author entropy to characterize the contributions of multiple authors to a given file, with an eye toward understanding the degree of collaboration and the most common interaction patterns.

1 Introduction

Software development is an inherently complex activity, often involving a high degree of collaboration between multiple individuals and teams, particularly when creating large software systems. Interactions between individual contributors can affect virtually all aspects of software development, including design, implementation, testing, maintenance, complexity, and quality.

Collaboration involves cooperation, communication, and coordination, and generally implies some governing organizational structure. The organization has an effect on the structure of the software being developed, as per "Conway's Law" [4]; presumably applying equally to proprietary and open source software. Brooks noted that potential communication channels increase as the square of the number of contributors [2]. Thus, there is benefit to understanding and managing collaboration so it does not become a liability.

Analyzing collaboration data can help explain how people work together to develop software. Studies by Bird [1], Ducheneaut [6], Gilbert [7], Mockus [12], Dinh-Trong [5], and others have examined interactions between open source developers by correlating communication records (such as email) with source code changes. Such approaches can expose patterns which reinforce contributor roles and module boundaries, but may not be feasible for all projects (particularly if email archives are unavailable) and can be difficult to compare or aggregate across disparate projects.

In addition to examining collaboration across projects and modules, there is value in understanding how contributors collaborate *within* files. Having a sense

of what constitutes "typical" collaboration for a project can provide valuable context. For example, if most files in a project have one or two authors, a file with 10 authors may merit additional scrutiny. In open source projects, unorganized and organic contributions may be evidence of the bazaar rather than the cathedral [13]. In any case, simply knowing can help set expectations.

This paper both replicates and extends earlier results [15]. Our research goals center around detecting, characterizing, and understanding patterns of collaboration within source code files. Our primary research questions are:

1. *How often do* n *authors contribute to a given file?*
   We anticipate that most files have a single author, and as the number of authors increases, the count of files with that many authors will decrease.
2. *Is there a higher degree of collaboration in small or large files?*
   We anticipate that there will be a positive correlation between file size and author count, partially because larger files have more code, and the potential for more distinct functionalities and individual responsibilities.
3. *Do files contain similar proportions of contributions from each author, or is there a dominant author who is the clear "owner" of a given file, and if so, how dominant is that author?*
   We anticipate that most source files will have one author who contributes significantly more code than any other single author, and that this author's dominance will be inversely related to the number of contributing authors.
4. *Is there a uniform or uneven distribution of collaboration across projects?*
   We anticipate that there will be a few "core" projects which are highly collaborative, and many ancillary projects which are less collaborative.

2 Methodology

We conducted an observational study on existing Eclipse projects by extracting author attribution data for Java source code files from git repositories. In this section we describe the process we used to select and obtain the data.

2.1 Project and File Selection

We chose to analyze Eclipse Foundation projects for several reasons, including:

- the number and variety of Eclipse-related projects,
- use of easily-recognizable programming languages,
- the ability to locally clone remote git repositories,
- a track record of sustained development activity,
- the existence of corporate-sponsored open source development projects.

We selected Java source files for our analysis, since over 92% of the source files in the repositories are Java, and Eclipse is so closely aligned with Java. We mined data from 251,633 files in 592 projects. We included auto-generated code in our analysis, since the inclusion of such files allows us to accurately characterize the state of the project to which they belong.

2.2 Extraction and Calculation

The first step in calculating author collaboration is to count how many authors have contributed to a file and the number of lines contributed by each one. Summarizing raw line counts with a single representative statistic per file allows for detailed statistical analysis of collaboration trends. In this paper, we use: (1) the percentage of lines attributed to the most dominant author in each file, and (2) author entropy (see Section 3 for details). These numbers can help characterize some aspects of author contribution patterns.

We created a bash script to locally clone each remote git repository and use 'git blame' to count the number of lines attributed to each author for each matching file. For each file in a project, the file path and line counts attributed to each author were recorded.

We then wrote a simple CLI tool to process this data and calculate the percentage of lines written by each author. Author entropy for each file was calculated using Equation 1. We also normalized entropy by dividing by the maximum possible entropy for each file, shown in Equation 2.

2.3 Limitations of the Data

We draw data only from git, a source control management (SCM) system that preserves snapshots of file state over time. We do not consider other collaboration mechanisms, such as email archives, forums, etc., although this could be a very interesting extension of this work.

It it important to note that the SCM record of who "owns" a line of code only identifies the individual who committed the most recent change affecting that line. It does not guarantee that the contributor actually conceived of, wrote, or even understands the code. By itself, it also does not tell us the genesis of a line; it could be new, a minor tweak, or a formatting change.

Because we consider only the latest revision of each file, this data cannot be used to make any inferences about collaboration over time. Without historical data, we can see the result of collaboration, but not the nature of the evolution of such collaboration.

Lastly, because we record author counts but not relative ordering of contributions from various authors, this data does not fully capture or express the amount of disorder. Because only percentages by each author are considered, the data makes no distinction between files with orderly, segregated blocks of contributions and files in which authors' contributions are all mixed together.

3 Author Entropy

Author entropy is a summary statistic that quantifies the mixture of authors' contributions to a file. Contribution percentages are weighted using logarithms

and summed; the resulting value conveys more information about the distribution than a simple average, and can expose interesting authorship patterns more readily than raw line counts. Taylor et al [15] introduced author entropy and examined distributions in a proof-of-concept study with SourceForge data. A follow-on paper [3] examined author entropy in GNOME application source.

Entropy originated in the field of thermodynamics, which defines it as the disorder or randomness of molecules in a system. Entropy has also been defined in terms of probability theory and used in the fields of information theory [14] and machine learning [11].

We apply entropy as a measure of collaboration between individual contributors. Specifically, we consider entropy of source code by counting the number of lines attributed to each author. This definition of entropy allows us to quantify the mixture of author contributions to a file.

3.1 Calculating Entropy

Entropy formulae are nearly identical across domains, and generally vary only in symbolic representation and constant multipliers. We use a formulation very similar to that used in machine learning.

If F is a file, A is the number of authors, and p_i is the proportion of the text attributed to author i, then the entropy of the file is defined as:

$$E(F) \equiv -\sum_{i=1}^{A}(p_i \cdot log_2\ p_i) \tag{1}$$

$E(F)$ is maximized when all authors contributed equal proportions of text in a file ($\forall\ i, p_i = \frac{1}{A}$). The upper limit of $E(F)$ is a function of A:

$$E_{max}(F) \equiv log_2\ A \tag{2}$$

We use log_2 for historical reasons tied to information theory (essentially, calculating the number of bits required to encode information). Although any logarithmic base would suffice, it is convenient that using log_2 results in entropy values in the range (0,1] for a binary classification.

3.2 Normalizing Entropy

Because the maximum possible entropy for a file is a function of the number of authors, intuitive understanding of entropy can be difficult. For example, an entropy value of 1.0 is the maximum possible for a file with 2 authors, but comparatively low for a file with 10 authors. Dividing E by E_{max} produces a normalized value in the range (0,1] which represents the percentage of maximum entropy. Normalized entropy can be easier to understand, and in some cases more appropriate for comparisons between disparate files.

4 Interpreting Collaboration

A high degree of collaboration within a given source file is not inherently good or bad; as with any metric, context is key. Without knowledge about additional factors such as a project's state, organization, and development conditions, interpreting collaboration is purely speculative. To illustrate this point, we list below a variety of factors that could influence author entropy.

Low entropy could result from factors as varied as:

- Well-architected and modular software.
- Excellent communication and coordination.
- Lack of involvement from potential contributors.
- A disciplined team in which each person "owns" a module.
- A gatekeeper who gets credit for code written by others.
- Code that few people understand.
- Code that was reformatted and old attributions lost.
- Code with exhaustive unit tests, known to be correct.
- Code with negligible unit tests and unknown defects.
- Auto-generated code that no human actually "wrote."
- Critical code that few people are allowed to modify.
- Mature code with little or no need for maintenance.
- Stale code that isn't touched, even if it needs fixing.
- Dead code which is no longer used or modified.

High entropy could result from factors as varied as:

- Code with high coupling or many inter-dependencies.
- Unrelated code entities being stored in a single file.
- Adding manpower to a late project (Brooks' law).
- Extremely buggy code that is constantly patched.
- Extremely stable code that is well-maintained.
- Enhancements or fixes that touch several files.
- Contributors joining or leaving a project team.
- Actively evolving code or refactoring activity.
- Miscommunication or lack of clear direction.
- Healthy collaboration between contributors.
- Overlapping responsibilities of contributors.
- Agile development or team programming.
- Potential for integration-stage problems.
- Continuous integration testing and fixes.

Such a menagerie of disparate factors is not a flaw in the metric itself, but rather suggests that any metric can easily be misinterpreted without proper context. For example, a file with high entropy written by several experts is likely of higher quality than a file written by one novice author. Two files may have similar entropies despite a large size difference. A recent contributor may

understand a file better than the original author who wrote it years ago. Correlating author entropy with other metrics and observations can help distinguish between "good" and "bad" entropy and provide valuable new insights.

Author entropy cannot directly indicate other attributes of the source code. For example, file length is obscured since files of different size but equal proportions of contribution have the same entropy. Entropy also does not reflect quality or the relative importance of contributions, such as new functionality, bug fixes, comments, whitespace, or formatting. Although different entropy calculation techniques could opt to account for such factors, there is no way to deduce the weighting of such factors from a single number.

5 Results

The line count of the source files we examined ranged from 1 to 228,089, with a median of 89. The extreme right-tail skew (97.5% have 1,000 lines or fewer, 92.5% have 500 or fewer) suggests that the data may have an exponential distribution. Plotting the data with a log_{10} transformation produces a histogram (Figure 1) that closely resembles a normal distribution. A Q-Q plot showed that the population fits a log-normal distribution quite well, although the long tail caused higher residual deviations in the upper range. We also examined the files with 10 lines or fewer and found that nearly all of them were related to unit tests; several projects have extensive tests with Java source files that specify in and out conditions, but have little or no code. Excluding these left-tail outliers greatly improved the fit of the Q-Q plot in the low range.

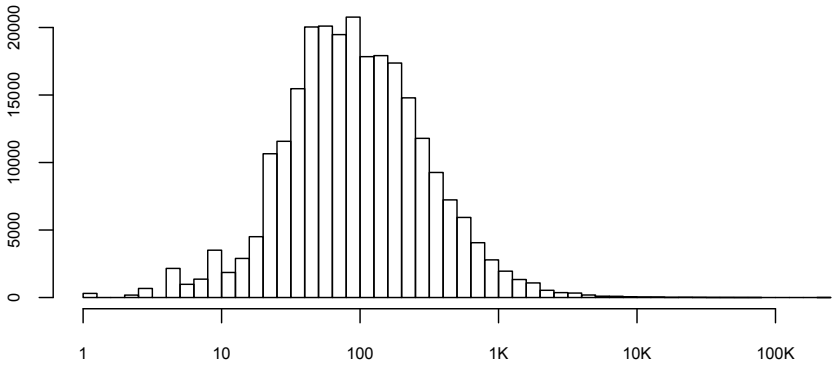

Fig. 1. Frequency of file sizes (in number of lines).

To answer our first research question, we plotted the frequencies of files with n authors. The resulting histogram was an exponential decay curve, and when plotted with a logarithmic scale, a near-perfect log-linear decay is evident (see Figure 2). This confirms our hypothesis that most files have a single author, and that the number of files with n authors decreases as n increases. It is also strikingly similar to Lotka's Law [10], which states that the number of

authors who contribute n scientific publications is about $1/n^a$ of those with one publication, where a is nearly always 2. Lotka's law predicts about 60% of authors publish only once; in our data, 58.22% of the files have one author.

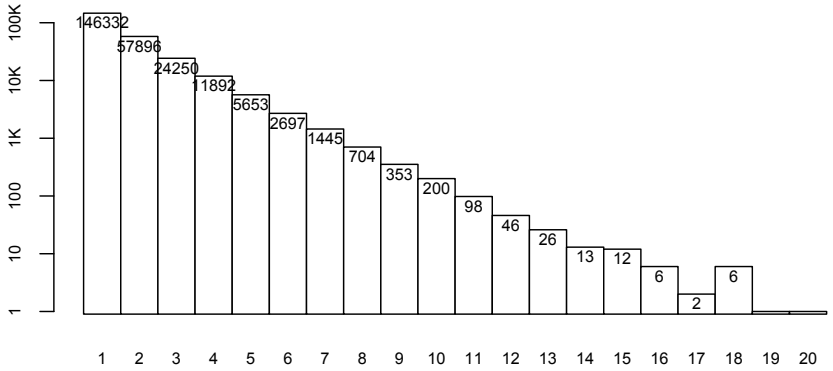

Fig. 2. Frequency of number of authors contributing to a given file.

To answer our second research question, we plotted file size distributions grouped by author count (see Figure 3). The log-linear increase in average file size as the number of authors increases confirms our hypothesis that, on average, there is more collaboration (i.e., more authors) in large files. However, we must note that there is a degree of uncertainty due to the decreasing sample sizes for higher author counts and the extreme outliers for lower author counts.

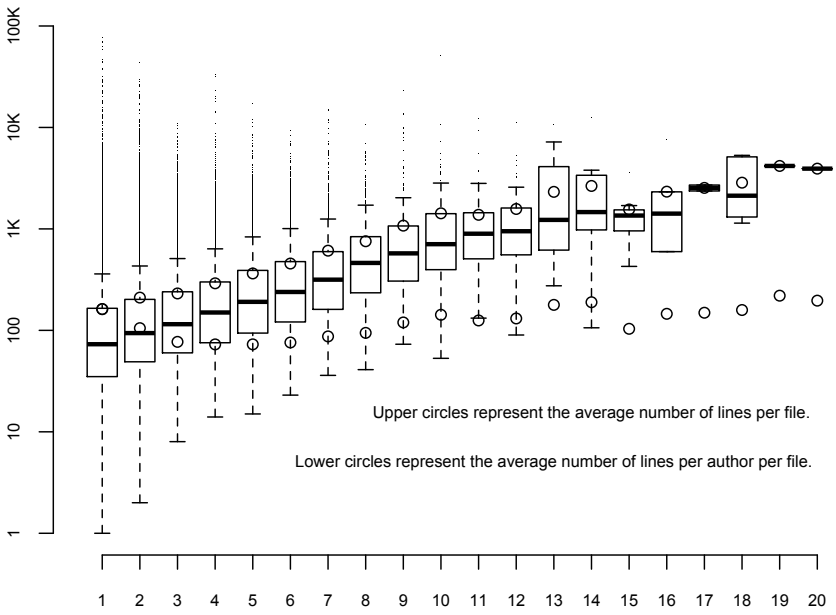

Fig. 3. Author count vs. file size (in number of lines).

We augmented Figure 3 with two additional data series: (1) the average number of lines in a file, and (2) the average number of lines contributed *per author* to a file. Note that there is a pronounced dip between 1 and 10 authors, but a fairly consistent average throughout. Although evaluating the causes and ramifications of this trend are beyond the scope of this paper, we find this to be an interesting topic for future work.

To answer our third research question, we plotted the number of lines in files with two or more authors against the percentage of lines attributed to the most dominant author in each file (see Figure 4). We also plotted the distributions of author dominance for all files with a given author count (see Figure 5).

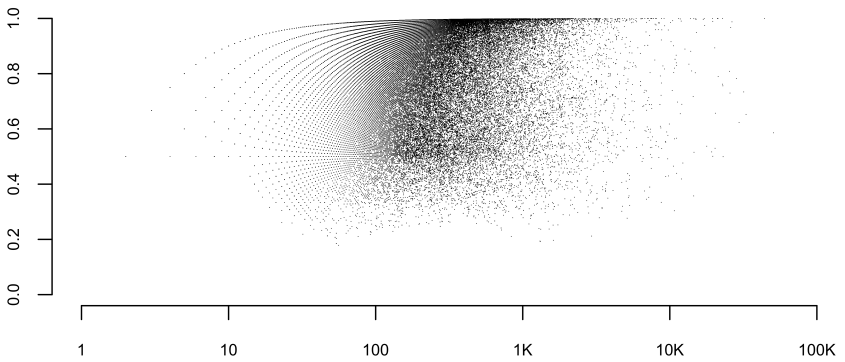

Fig. 4. Line count vs. percent written by dominant author for files with 2+ authors.

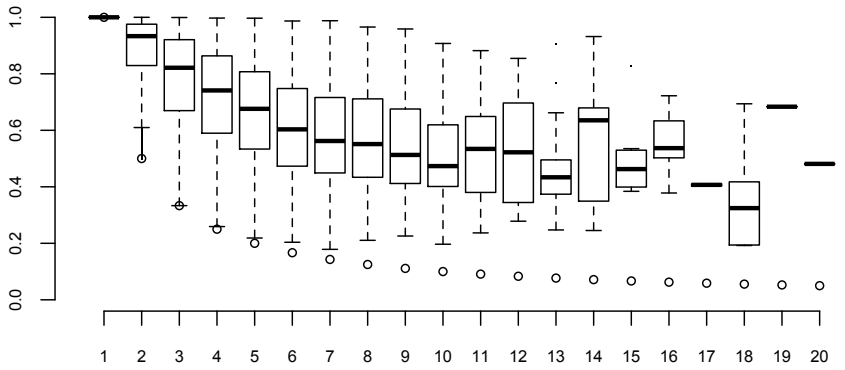

Fig. 5. Author count vs. author dominance. Circles represent the curve $\frac{1}{x}$.

These plots confirm our hypothesis that most files have a dominant author, and that the percentage of lines attributed to that author generally decreases as the count of contributing authors increases. We find it noteworthy that author dominance diverges increasingly from the lower bound ($\frac{1}{x}$). This suggests that in Eclipse projects, more authors contributing to a file does not imply balanced contributions; rather, a single author usually dominates the others.

To answer our fourth research question, we plotted the number of lines in a project against the number of unique authors contributing to it for all 592 projects (see Figure 6). Over 83% of the projects have 10 or fewer unique authors, and some significant outliers have much larger numbers of authors.

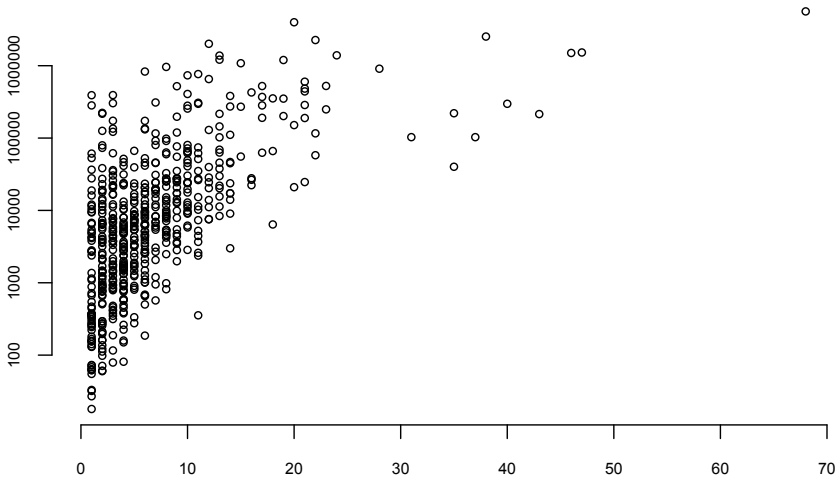

Fig. 6. Author count vs. total number of lines for all 592 projects.

We also manually examined the 211 files with 11 or more authors. Nearly all of these files came from a handful of projects, all of which were among the top 25 projects with the most authors. These projects include:

- org.eclipse.ui.workbench (Eclipse IDE interface)
- org.eclipse.jdt.core (Java Development Tools)
- org.eclipse.cdt (C/C++ Development Tooling)
- org.eclipse.pdt (PHP Development Tools)
- org.eclipse.birt.report (Business Intelligence and Reporting Tools)
- org.eclipse.jface (UI application framework/toolkit based on SWT)

The nature and role of these highly-collaborative projects confirms our hypothesis that collaboration is not distributed uniformly, but is concentrated in a few core projects. This phenomenon is also related to our second research question, about the relationship between collaboration and file size.

5.1 Additional Questions

In addition to our primary research questions, we also replicated some results from prior related work to verify whether the assertions made therein still hold for broader data. These results are related to distributions of author entropy (see Section 3) over varying file sizes and author counts.

In [15] we found a positive relationship between author count and entropy (entropy rises as the number of authors increases). We found the same trend in Eclipse source code, although it breaks down somewhat for 11 or more authors due to sparseness of data (see Figure 7).

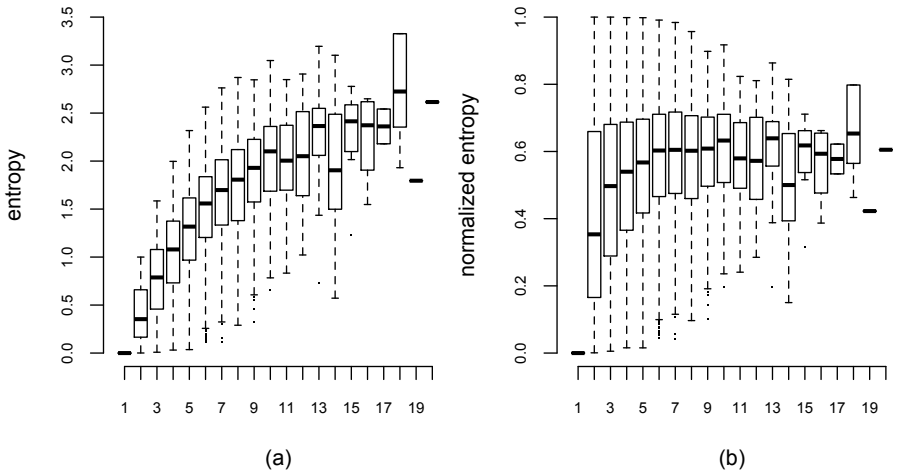

Fig. 7. Author count vs. (a) entropy and (b) normalized entropy.

Although the entropy metric is inherently biased toward higher values (particularly when there are more authors), any file can have low entropy when one of the authors is extremely dominant. However, because the maximum possible entropy for a given file is a function of the number of authors, it can be difficult to compare entropies for files with different number of authors. For this reason, we use normalized entropy, which always falls in the range [0,1] regardless of author count, and thus represents the percentage of maximum possible entropy.

Interestingly, the data exhibits a trend previously observed [15] in a very constrained set of SourceForge data: distributions of normalized entropy tend to center around 0.6 (or 60% of maximum possible entropy) as the author count increases. Even as the data becomes more sparse for higher author counts, the distributions tend to converge on the same range.

Casebolt [3] examined two-author source code files and observed an inverse relationship between file size and entropy (small files have high entropy and vice versa). A similar pattern occurs in our data, as shown in Figure 8(a). Unfortunately, it is impossible to discern how many data points share the same location. The task is even more hopeless when all files (not just those with two authors) are included in the same plot, as in Figure 8(b). To better understand the distribution and density of these data, we borrow a tool used by Krein [9] to visualize language entropy: 3D height maps. This technique generates an image in which densely-populated regions appear as elevated terrain (see Figure 9).

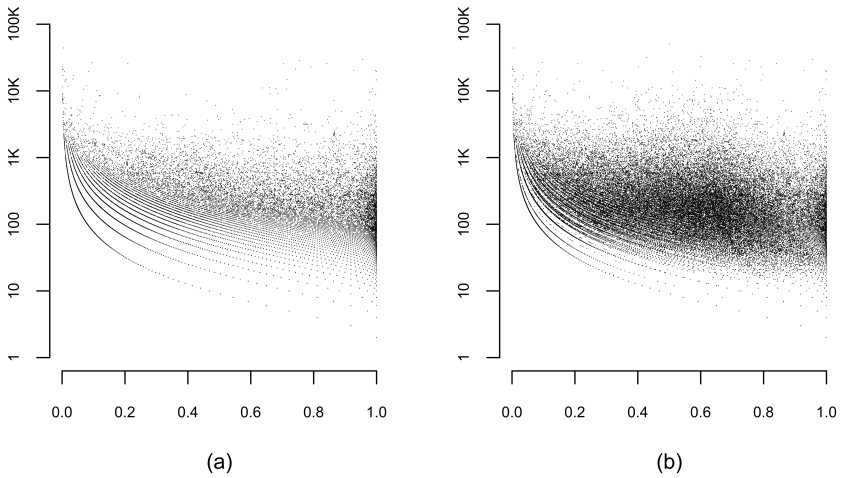

Fig. 8. Normalized entropy vs. line count for (a) two authors and (b) all files.

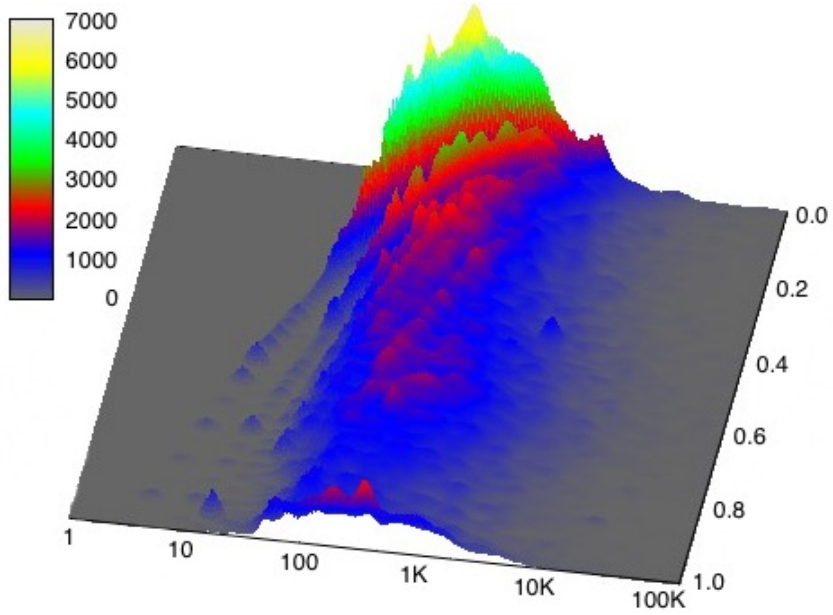

Fig. 9. Height map of line count vs. normalized entropy (same data as Figure 8b).

Figure 9 is an adaptation of both Figure 8(b) (adding height) and Figure 1 (adding depth). Starting from the back/left, the curves represent files in which one author "owns" all lines but one, two, etc. The disproportionate distribution of files on the furthest curves suggests that one- and two-line edits are probably extremely common occurrences in the projects we examined. This may be a manifestation of many small bug fixes, refactorings, interface changes, etc.

6 Future Work

Although this paper both replicates and adds to the results of prior work [15], we also see several promising ways to extend this research.

First, statistical analysis of author entropy over time, including how entropy of files, modules, and projects change over time, and why. One limitation of this paper is that we examine only the most recent version of each file; we do not consider previous versions of existing files, or files which once existed but have since been deleted. We see significant value in understanding not only code ownership, but the degree of the resulting disorder, and how it is related to and caused by the development processes at play within a project.

Second, correlation of entropy with other code ownership measurements, communication records (such as email), and development roles. This could build on studies such as those by Bird [1], Mockus [12], Dinh-Trong [5], Jensen [8], and von Krogh [16], among others. Understanding how conributor roles and project organization affect source code entropy could help OSS project administrators (or "core team") to more effectively analyze and organize contributors' efforts.

7 Conclusion

We discovered that author attribution data for source code files can provide insight into the nature of collaboration between source code contributors. As researchers, we seek to understand how people work together to develop complex systems, and to explain success or failure based on the data at our disposal. We are fascinated by the patterns of order which seem to naturally fall into place amid the organic chaos of free-form interactions.

Our study revealed similar authorship patterns in a vastly different code base than prior work, and suggested interesting new patterns we had not previously considered. Author entropy continues to be an interesting and useful metric for characterizing contributor interactions. Future research will improve our ability to link collaborative activity with the underlying factors that influence it, and facilitate improvements that enhance the quality of the software we produce.

References

1. 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*, November 2008.
2. F. P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley Longman Publishing Co., Inc., January 1975.
3. J. R. Casebolt, J. L. Krein, A. C. MacLean, C. D. Knutson, and D. P. Delorey. Author Entropy vs. File Size in the GNOME Suite of Applications. *Proceedings of the 6th IEEE Working Conference on Mining Software Repositories (MSR '09)*, pp. 91–94, May 2009.

4. M. E. Conway. How Do Committees Invent? *Datamation*, 14(4):28–31, April 1968.
5. T. T. Dinh-Trong and J. M. Bieman. The FreeBSD Project: A Replication Case Stdy of Open Source Development. *IEEE Transactions of Software Engineering*, 31(6):481–494, June 2005.
6. N. Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. In *Computer Supported Cooperative Work*, volume 14, pp. 323–368, 2005.
7. 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. University of Illinois, Springer, September 2007.
8. C. Jensen and W. Scacchi. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In *29th International Conference on Software Engineering*, ICSE '07, pp. 364–374, May 2007.
9. J. L. Krein, A. C. MacLean, C. D. Knutson, D. P. Delorey, and D. L. Eggett. Impact of Programming Language Fragmentation on Developer Productivity: a SourceForge Empirical Study. *International Journal of Open Source Software and Processes (IJOSSP)*, 2(2):41–61, June 2010.
10. A. J. Lotka. The frequency distribution of scientific productivity. *Journal of the Washington Academy of Sciences*, 16(12):317–324, June 1926.
11. T. M. Mitchell. *Machine Learning*, pp. 55–57. McGraw-Hill, 1997.
12. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, July 2002.
13. E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly and Associates, Inc., 2001.
14. C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.
15. Q. C. Taylor, J. E. Stevenson, D. P. Delorey, and C. D. Knutson. Author Entropy: A Metric for Characterization of Software Authorship Patterns. In *Proceedings of the 3rd International Workshop on Public Data about Software Development (WoPDaSD '08)*, September 2008.
16. G. von Krogh, S. Spacth, and K. R. Lakhani. Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. *Research Policy*, 32(7):1217–1241, July 2003. Open Source Software Development.