# The Commit Size Distribution of Open Source Software

Oliver Arafat, Dirk Riehle
SAP Research, SAP Labs LLC
3410 Hillview Ave, Palo Alto, CA, 94304 U.S.A.
oarafat@gmail.com, dirk@riehle.org

## Abstract

*With the growing economic importance of open source, we need to improve our understanding of how open source software development processes work. The analysis of code contributions to open source projects is an important part of such research. In this paper we analyze the size of code contributions to more than 9,000 open source projects. We review the total distribution and distinguish three categories of code contributions using a size-based heuristic: single focused commits, aggregate team contributions, and repository refactorings. We find that both the overall distribution and the individual categories follow a power law. We also suggest that distinguishing these commit categories by size will benefit future analyses.*

## 1. Introduction

Over the last 10 years, open source software has become an important cornerstone of the software industry. Commercial users use open source software as stand-alone applications and software vendors embed it into their products. Surprisingly, from a commercial perspective, open source software is being developed in ways that are different from the way corporations typically develop software. Research into open source best practices illustrates the desire to understand how open source software development works [1]. Once understood, corporations could learn from open source processes and adapt best practices to their needs.

The public nature of open source software development makes a quantitative approach to analyzing open source software development processes possible. Data about the actual behavior of software developers is readily available to researchers in source code repositories, in mailing list archives, and on project websites. This is in sharp contrast to closed source projects, which typically remain hidden behind corporate firewalls. As a consequence, much effort is being spent on mining open source code repositories, as for example the workshop series on "mining software repositories" shows [2] [3] [4] [5] [6].

Software development consists of making code contributions, also known as commits, to a source code repository that hosts the project. A particular challenge for analyzing software developer behavior and better understanding the open source software development process is to understand the intent of these code contributions. In particular, any of these commits may have a different intent [9] [10] [15]. Some commits fix bugs; others provide new features. Some remove code to clean up a project; others add whole new libraries in one go. Some commits are the result of a single development step by a single developer; other commits are the result of a whole team providing a year's worth of work in the form of a new component.

In this paper we define three main categories of commits. We present a size-based heuristic for determining into which category a commit falls. We base our discussion and conclusions on the analysis of the commit histories of 9,363 open source projects, providing more than 8 million commits. The main contribution of this paper is

- to present the overall commit size distribution on a level of scale that has not been done before,
- to split it up into three main parts, and to show that these distributions all follow a power law.

Section 2 introduces our database and approach, Section 3 presents the analysis of the commit size distribution, Section 4 discusses the limitations of our approach, Section 5 presents related work, and Section 6 discusses some final conclusions.

## 2. Data Source and Approach

We use the database of the open source analytics firm Ohloh Inc. [13]. Ohloh provides information from project websites and source code repositories. The data is stored in a relational database and is available to the interested public [16]. The database contains high-level data, such as the name of a project and its developers as well as fine-grain data like every individual code contribution ever made to the project. We only use data that was derived from the original project websites and ignore data available from Ohloh only.

Initially, Ohloh seeded its database with the 5,000 most popular open source projects. After this, Ohloh opened up its service for community editing in 2006. By March 2008, another 6,000 projects had been added by the community. This article is based on a March 2008 database snapshot, which contains 9,363 completely crawled and analyzed projects covering a time frame from January 1990 to February 2008.

The Ohloh database provides the complete configuration management history of each crawled project (to the extent available on the web). Thus, every single commit action of all the projects over their entire history is available. A commit is the action with which a developer contributes a piece of code to the project's repository. A commit consists of a set of file modifications that may encompass any of the following three actions: the addition of code, the removal of code, and the changing of existing code.

We measure the size of commits in this paper in source lines of code (SLoC) using Ohloh's own open source diff tool [18]. SLoC consist only of actual program code, omitting empty and comment lines. For the purposes of this paper, we assume that one line of source code added constitutes about the same amount of work as one line of code removed or one line of code changed. Thus, we add up the number of lines added, removed, or changed to calculate the size of a given commit.

A commit consists of several diffs each affecting one file. For a given diff, the database provides the following raw data:

- number of source lines of code added, and
- number of source lines of code removed.

We call one source code line of code added, removed, or changed "one modification". A commit typically includes multiple modifications.

Unfortunately, a changed line of code is counted as one line added and one line removed. This is a common problem in source code analytics, as there is no certain way of determining whether a changed source line of code isn't really one line removed and a new one added (short of asking the developer).

For example, the tuple (1 SLoC added, 1 SLoC removed) might represent any of the two commit events displayed in Table 1. Similarly, the tuple (4 SLoC added, 3 SLoC removed) might represent any of the four commit events displayed in Table 2.

Let $a_c$ be the number of SLoC added in commit c, and $r_c$ be the number of SLoC removed in commit c. We can then define the lower and upper bound for the number of modifications in the commit:

- lower bound: $lb_c = \max(a_c, r_c)$
- upper bound: $ub_c = a_c + r_c$

The probabilities for modification sizes are unevenly distributed. Thus, an assumption of equal distribution does not hold in general. By using a subset of our data and the changed line heuristic of the diff tool (based on solving the Longest Common Subsequence problem) we calculated the probability distribution for any such modifications [17]. In this paper, we use this distribution for calculating the size of the individual commits as we add them up to derive the overall commit size distribution of our project population.

Table 1: Interpretation of the database entry 1 SLoC added, 1 SLoC removed.

| **(1, 1)** | Number of SLoC added | Number of SLoC removed | Number of SLoC changed | **Number of Modifications** |
|---|---|---|---|---|
| **Commit event 1** | 0 | 0 | 1 | **1** |
| **Commit event 2** | 1 | 1 | 0 | **2** |

Table 2: Interpretation of the database entry 4 SLoC added, 3 SLoC removed.

| **(4, 3)** | Number of SLoC added | Number of SLoC removed | Number of SLoC changed | **Number of Modifications** |
|---|---|---|---|---|
| **Commit event 1** | 1 | 0 | 3 | **4** |
| **Commit event 2** | 2 | 1 | 2 | **5** |
| **Commit event 3** | 3 | 2 | 1 | **6** |
| **Commit event 4** | 4 | 3 | 0 | **7** |

## 3. Analysis and Discussion

In total, our database snapshot encompasses 8,556,036 commits. Of these, 215,531 commits have a size of zero. A commit size of zero occurs if no source lines of code are modified and only empty or commented lines of code are affected. In addition, 22,429 commits only move files from one location to another. With our focus on source code analysis we omit these commits for the rest of the analysis. This results in a total of 8,318,076 commits that are taken into consideration for further analysis.

Figure 1 shows the total commit size distribution of our sample population on a log/log-scale. The commit sizes range from 1 modification to 7,881,674 modifications. We used logarithmic binning to reduce the raw data to a data set more easily manageable for display in Figure 1.

### 3.1 Total Commit Size Distribution

Prior work has categorized commits by semantic intent, for example, whether a bug was fixed or a new feature was added, see Section 5. In this paper, we are interested in reviewing commits by size, so we decided to distinguish the following three main categories:

- *Single individual developer contribution (single commit).* In this case, a software developer makes a code contribution that (ideally) deals with exactly one semantic issue, for example, fixes a bug or works towards contributing a feature.
- *Aggregate developer contribution (aggregate commit).* In this case, a developer or a team of developers contributes a consolidated set of commits they had been working on separately, for example in another repository. An aggregate commit typically consists of multiple single commits.
- *Component or repository refactoring or consolidation.* In this case, a typically large commit resulted from integrating a whole library or branching a project.

Without prior knowledge, there is no certain way of determining into which of these mutually exclusive categories a given commit falls. However, for the analysis of behavior of open source software developers we would like to distinguish these commits from each other.

We suggest distinguishing commit types by their size, using the following simple heuristic:

- single commits—1 to 100 SLoC,
- aggregate commits—101 to 10000 SLoC, and
- repository refactorings—more than 10000 SLoC.

The following discussion is structured along the lines of these three categories.

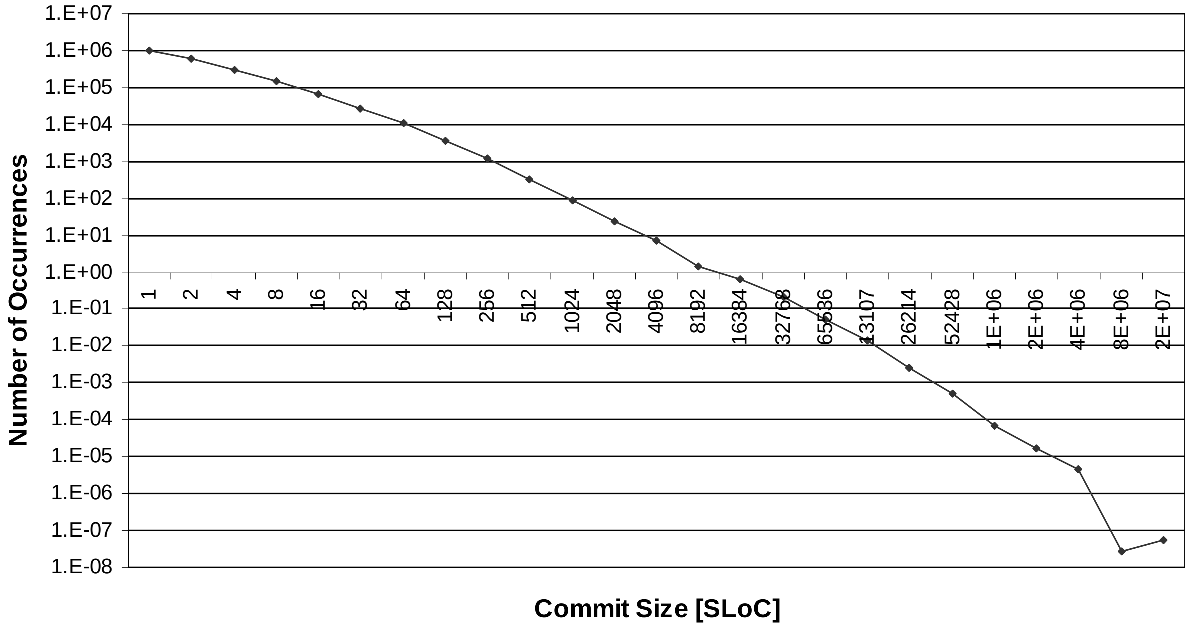

Figure 1: The total distribution of commit sizes after logarithmic binning (log/log scale).

Table 3: Models for single commits.

| Model | SLoC Range | Function | R-square |
|---|---|---|---|
| 1 | 1-100 | $y = 2E{+}06x^{-1.1326}$ | 0.9895 |
| 2 | 11-100 | $y = 3E{+}06x^{-1.2464}$ | 0.9971 |
| where x = commit size in SLoC and y = number of occurrences of the commit size | | | |

## 3.2 Single Commits

Figure 2 shows the distribution of commit sizes for single commits, defined as commits with less than or equal to 100 commits, on a log/log scale. A power relationship based curve has been fitted, providing an $R^2$ value of 0.9895. The good fit with the model suggests that single commits follow a power law.

These commits in the range of 1-100 SLoC constitute 83.54% of our total commit population. One-liners constitute 12.13% of the total sample population, two-liners constitute 8.964% of the population, three-liners constitute 5.449%, and so on. Figure 3 displays the percentage (of all commits) of commit sizes 1 to 10. The smaller the size of a commit, the more likely it is.

The dominance of small commits and their (almost strictly) falling probabilities as commit sizes get larger is somewhat surprising. This may be a result of the specifics of open source software development processes where many contributors cannot directly commit code but rather have to channel it through a committer [19]. This may have led to the dominance of small commits; a comparison with closed source software development is on our agenda for future work.

Table 3 shows a power relationship based model for the data in Figure 2. It adds a second model for the SLoC range of 11-100 SLoC, omitting the ten smallest commit sizes. The $R^2$ value for this second model is even better than for the first model, suggesting that the smallest commit sizes may involve their own types of commit, warranting further investigation.

## 3.3 Aggregate Commits

We distinguish single commits, which represent focused contributions like fixing a particular bug or implementing a feature, from aggregate commits, which represent larger aggregated chunks of work. Such aggregate commits may be the result of

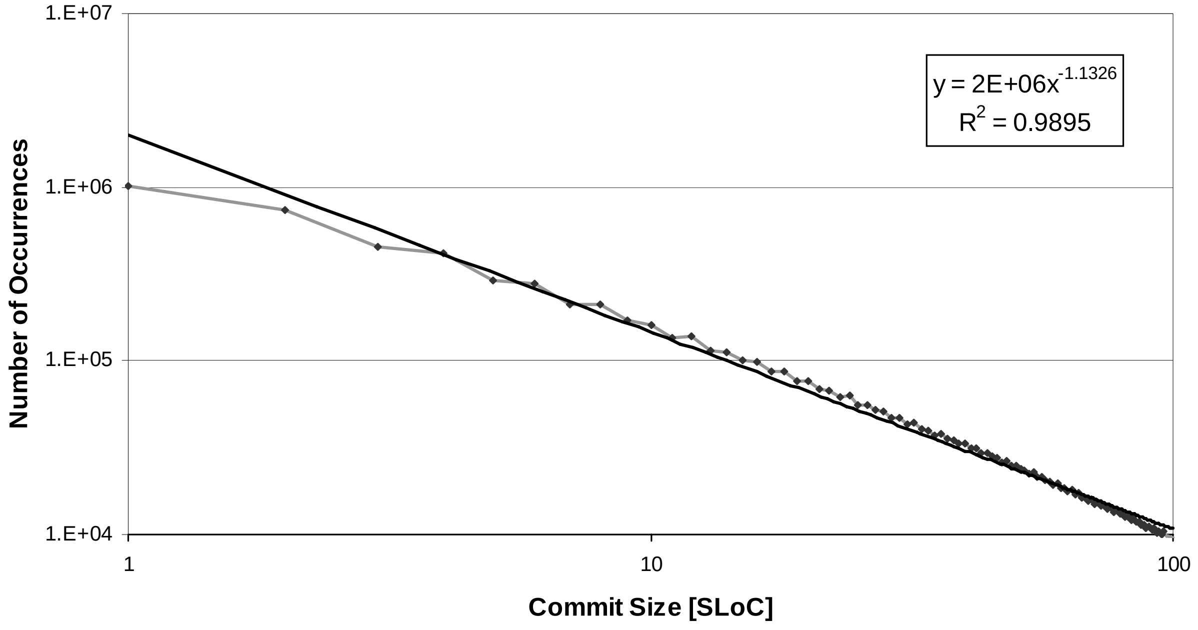

Figure 2: The distribution of commit sizes in the range 1 to 100 SLoC (log/log scale).

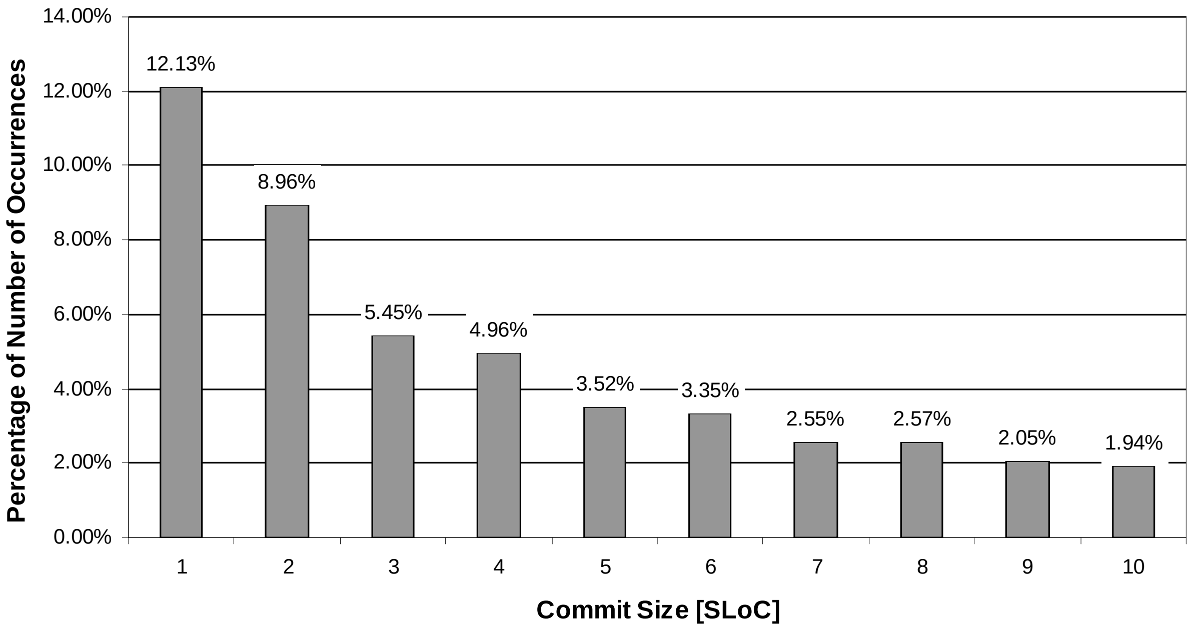

Figure 3: Absolute percentage of 1-10 SLoC commit sizes of total sample population.

- a single developer or a team of developers delaying commits to finish a complex feature, or
- a development process using distributed configuration management systems.

Other explanations are possible as well. The result is the same: The commit is a collection of prior commits, typically single commits made separately from the main code repository. The key distinguishing fea-

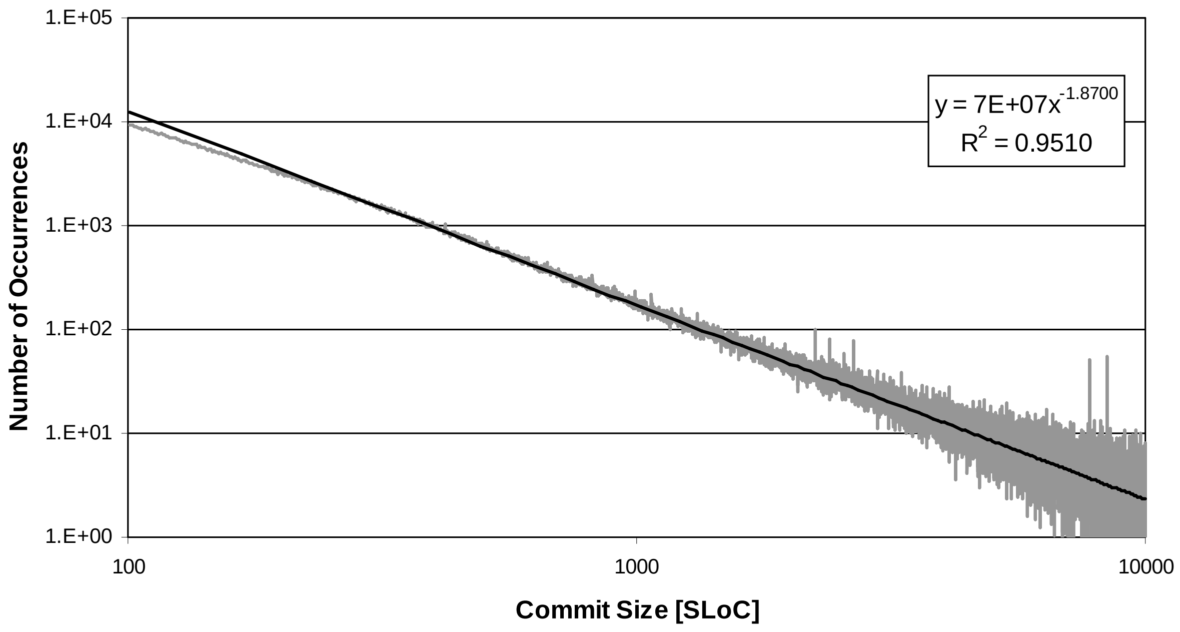

Figure 4: Distribution of commit sizes in the range of 101-10,000 SLoC (log/log scale).

ture from single commits is the size, and we suggest that size to cover the range of 101-10,000 SLoC.

Figure 4 shows the number of occurrences of commit sizes in the range of 101-10,000 SLoC. When fitting a curve using a power relationship based model to Figure 4, we get an $R^2$ value of 0.9510, suggesting that this section of the overall commit size distribution also follows a power law.

## 3.4 Repository Refactorings

Figure 5 shows the remaining commit occurrences for sizes above 10,000 SLoC. This Figure uses an equal distribution of commit size probabilities approximation, cf. [17]. We intend to refine the model in future work.

Such large commits may be the result of

- a large copy and paste, for example the inclusion of an existing library, or
- the initial check-in or branching or forking of an existing project, or
- the merging of separate repositories in distributed configuration management system.

These large commits represent aggregate commits as well. Fitting a power relationship leads to an approximate model with an $R^2$ of 0.7025. We attribute the limited goodness of fit to the sparseness of data and our equal distribution approximation.

Table 4: Model for total distribution.

| |
|---|
| $y = 1E{+}07x^{-1.8612}$ |
| $R^2 = 0.9782$ |
| where<br>x = commit size in SLoC and<br>y = number of occurrences of the commit size |

## 3.5 Fitting to Power Law Models

Fitting a power relationship for Figure 1, the total sample population (after logarithmic binning), provides the model and $R^2$ shown in Table 4. The high $R^2$ for the overall distribution suggests once more that it follows a power law. The individual sections discussed in this paper have an equal or even better fit, though this is not surprising given the constrained range. However, we take the overall goodness of fit as a strong indicator that a power relationship based model is a good approximation of the overall distribution.

Given the frequent use of visualizations in this paper, one might suspect that we derived the power-law model for the commit size distribution using graphical interpolation for linear functions on the log/log-scale graph. This has been shown to lead to inaccurate results [14]. In contrast to such approaches, we used least-square fitting for determining the best matching

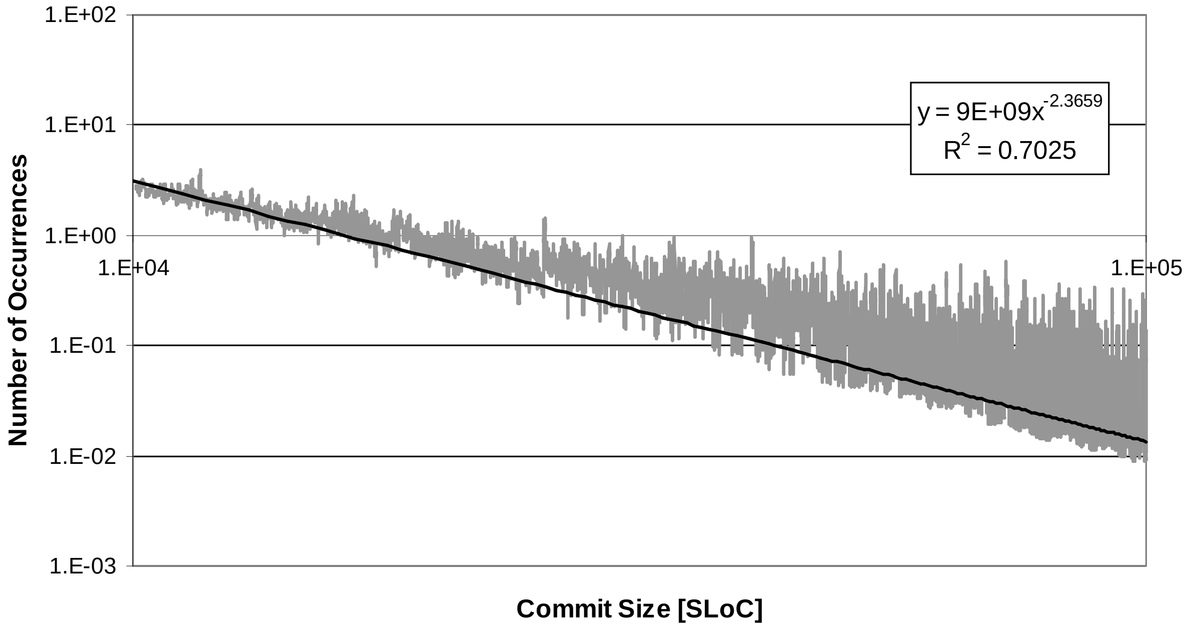

Figure 5: Distribution of commit sizes above 10,000 SLoC (approximation, log/log scale)

function for the raw distribution data. Goodness of fit, i.e. $R^2$, is calculated as the coefficient of determination, here specifically the Pearson Correlation Coefficient, as provided by the worksheet function of Microsoft Excel. Given the amount of data and the goodness and smoothness of fit, we are confident in confirming the power-law relationships we found.

## 4. Strengths and Limitations

A key strength of the presented work is the large sample size. The projects represented in our database draw broadly across all of open source, with no apparent bias towards any particular source. If any, the bias is towards current and popular projects. As mentioned, Ohloh's database was seeded with the top 5000 popular open source projects using Yahoo! search engine in-links as the main measure. We had no control over this algorithm. The subsequent enlistment of further projects was done by the community which given the young age of the Ohloh service suggests a bias towards current popular open source projects rather than dead ones. Thus, we feel confident in claiming that our results are representative for the current state of open source.

The cut-off values we provide to categorize commits into three different distinct categories represent a simple heuristic only. We gained some confidence in these values from the observation that each individual segment seems to be best explained by a power law. We concur with Madey et al. who observed that power laws dominate many relationships we can find in peer production based systems like open source [8].

Still, using size as the classification criterion only will inadvertently miss-classify many commits. For example, it is safe to assume that there will be many single commits with more than 100 SLoC. Our intent here is to demonstrate a pragmatic filter for further open source analysis and to explore its properties. Whether our heuristic is appropriate or not will depend on the particular analysis being undertaken.

Also, while our three-category classification is intuitively appealing, not all software developers behave in such clean-cut ways, and there may be many commits that mix purposes, for example, adding an external component, fixing a bug, and implementing a new feature all in one commit.

Maybe most significantly, our cut-off values of 100 SLoC and 10,000 SLoC are more based on observation and intuition and less based on statistical analysis. The refinement of this heuristic will be a next step to be undertaken, perhaps using discriminant analysis to determine models for the different categories and finding optimal cut-off values to distinguish the categories from each other.

## 5. Related Work

Purushothaman and Perry discuss the impact of small changes on software projects [9]. They observe that during maintenance, nearly 10% of all changes made to the project under investigation were one-line changes. This is close to our finding in Section 3.2 that one-liners constitute slightly more than 12% of all commits.

Hindle et al. analyzed nine open source projects for the size of their commits, distinguishing small from large commits [10]. They found that small commits are more corrective while large commits are more perfective. Finally, they classified large commits by intent, creating more than 20 categories for large commits. Their notion of large commit is based on the number of files touched and not of the number of source lines of code involved, and hence is hard to compare with our approach. In general, the types of large commits they found seem similar to our notion of large component or repository refactoring or consolidation.

Mockus and Votta looked at reasons for software changes in the version histories of two closed source projects [15]. The focus of their work was deriving the reasons of changes from the commit comments. In the process of their research, they (re-)discovered the three well-known types of adaptive, corrective, and perfective changes. Of these changes, they found that corrective changes ("bug fixes") are the smallest. All of these maintenance activities led to comparatively small changes and would mostly correspond to what we identified as single commits in this paper.

Some amount of work has been spent on more accurately tracking actual changes in software projects. As noted, most configuration management systems only let you track the number of lines added and removed and loose the information whether a line was changed and how. Canfora et al. have developed a distance metric and algorithm for determining code similarities between two source file revisions [11]. However, such work represents a statistical measure and not a certain way of determining a source line of code change over a combined removal/addition. Moreover, it is computationally so expensive that we did not consider it for our analysis. Robbes enhanced an IDE to track the changes and a configuration management system to store them [12]. Unfortunately, this system is not in widespread use and data was not available.

## 6. Conclusions

In this article, we show how code contributions to source code repositories in open source software development follow a power-law distribution and can be split into three distinct categories. We provide a simple size-based heuristic for distinguishing between the three categories. In future work we intend to improve the statistical analysis on the various aspects presented. For example, we intend to use discriminant analysis to further distinguish commit types and to improve our SLoC-based heuristic. Such distinct types may stem from different development processes, different phases in the process, differences in tooling, etc. Finally, we are working on comparing these and other properties of open source software with data gathered from closed source code repositories.

## Acknowledgements

We would like to thank Amit Deshpande, James Noble, Ewan Tempero as well as the anonymous reviewers for helping us improve this paper through their constructive feedback.

## References

[1] Walt Scacchi. "Free/Open Source Software Development: Recent Research Results and Emerging Opportunities." In *Proceedings of ESEC/FSE 2007.* ACM Press, 2007.

[2] MSR 1. *Proceedings of the 1st International Workshop on Mining Software Repositories*. IEEE Press, 2004.

[3] MSR 2. *Proceedings of the 2st International Workshop on Mining Software Repositories.* IEEE Press, 2005.

[4] MSR 3. *Proceedings of the 3rd International Workshop on Mining Software Repositories.* IEEE Press, 2006.

[5] MSR 4. *Proceedings of the 4th International Workshop on Mining Software Repositories.* IEEE Press, 2007.

[6] MSR 5. *Proceedings of the 5th International Workshop on Mining Software Repositories.* IEEE Press, 2008.

[7] James Howison and Kevin Crowston. "The Perils and Pitfalls of Mining SourceForge" In *Proceedings of MSR 1,* see [2].

[8] Gregory Madey, Vincent Freeh, Renee Tynan. "Modeling the Free/Open Source Software Community: A Quantitative Investigation" In *Free/Open Source Software Development,* ed., Stephan Koch, Idea Publishing, 2004.

[9] Ranjith Purushothaman and Dewayne E. Perry. "Towards Understanding the Rhetoric of Small Changes." In *Proceedings of MSR 1,* see [2].

[10] Abram Hindle, Daniel M. German, and Ric Holt. "What Do Large Commits Tell Us? A taxonomical study of large commits." In *Proceedings of MSR 5,* see [6].

[11] Gerardo Canfora, Luigi Cerulo, Massimiliano Di Penta. "Identifying Changed Source Code Lines from Version Repositories." In *Proceedings of MSR 4,* see [5].

[12] Romain Robbes. "Mining a Change-Based Software Repository." In *Proceedings of MSR 4,* see [5].

[13] Ohloh, Inc. See http://www.ohloh.net.

[14] Michel L. Goldstein, Steven A. Morris, Gary G. Yen. “Problems with Fitting to the Power-Law Distribution.” See http://arxiv.org/abs/cond-mat/0402322.

[15] Audris Mockus, Lawrence G. Votta. “Identifying Reasons for Software Changes Using Historic Databases.” In *Proceedings of the 2000 International Conference on Software Maintenance* (ICSM 2000). IEEE Press, 2000. Page: 120-130.

[16] Ohloh, Inc. Ohloh API. See http://www.ohloh.net/api.

[17] Philipp Hofmann, Dirk Riehle. “A Statistic for Calculating Commit Size Probabilities in Open Source Projects.” Technical Report, forthcoming.

[18] Ohloh, Inc. ohcount. See http://labs.ohloh.net/ohcount.

[19] Dirk Riehle. “The Economic Motivation of Open Source: Stakeholder Perspectives.” *IEEE Computer*, vol. 40, no. 4 (April 2007). Page 25-32.