Understanding Change Contribution Patterns in Open Source and Commercial Software Projects

Jai Asundi, Rick Kazman, V. S. Arunachalam

CAPRI Publication 05-05

Jai Asundi is on the faculty of the School of Management at the University of Texas at Dallas. Rick Kazman is in the College of Business Administration, University of Hawaii. V. S. Arunachalam is Center for the Study of Science, Technology and Policy, India.
Understanding Change Contribution Patterns in Open Source and Commercial Software Projects

Jai Asundi, School of Management, University of Texas at Dallas, Richardson, TX 75080.
asundi@utdallas.edu

Rick Kazman, Department of Information Technology Management, College of Business Administration, University of Hawaii, HI 96822, kazman@hawaii.edu

V. S. Arunachalam, Center for the Study of Science, Technology and Policy (CSTEP), Bangalore 560078, India, vsarun@gmail.com

Please do not distribute without the authors’ explicit permission

Acknowledgments: The authors would like to thank Ashish Arora, Rajiv Jayant, Stan Liebowitz, Syam Menon, Audris Mockus and Sumit Sarkar for their invaluable comments, research assistance, data and suggestions for this paper. We also thank the Center for the Analysis of Property Rights and Innovation at the University of Texas at Dallas for research support.
0. ABSTRACT

Despite the substantial interest in and considerable market impact of Open Source Software (OSS), there have been few empirical studies that rigorously describe or analyze this form of development. In this paper, we analyze the OSS development process, and in particular the manner in which contributions are made to the OSS product. We used detailed information from the contribution logs of two OSS projects as a basis for our examination of this software development model. Our empirical analysis indicates that in spite of a relatively large number of people participating in the OSS projects, a majority of the code-related contributions come from a select few developers. Using the Gini coefficient to compare and contrast contribution patterns within the OSS projects and four commercial projects, we find that the distribution of software development is relatively more concentrated in the OSS projects. We then investigate the market conditions that might explain these findings. Our findings call into question the numerous claims made in the practitioner press that there are “thousands of eager co-developers” sifting through the code and making corrections. Instead, our results substantiate the general principle that the majority of software development activities are carried out by a small, cohesive group of individuals. The implications of our research are important not only for our understanding of the management and organization of open source software efforts, but also for our understanding of the corporate business and software development strategies that are likely to emerge during dislocations caused by the phenomenon of OSS development.
1. INTRODUCTION

Although the term “Open-source software” (OSS) was coined in 1998 by a group of software professionals working on projects where they freely distributed the source code along with the object-code to users, the sharing of source code has far longer lineage. An early example of sharing source code can be found in the distribution of the UNIX operating system source code to users that took place in the 1970s (McKusick, 1999). However, most early cases had no formal structures to incorporate and disseminate the changes proposed by various users. Changes were made in an informal fashion and most of the distributions were disseminated on tape disks using surface mail.

In the past decade there has been a surge of interest in OSS programs (e.g. Linux, Perl Apache, and FreeBSD). These efforts are often seen as a response to the products from large commercial firms, particularly those with dominant market positions such as Microsoft. With the apparent success of some OSS projects, organizations developing commercial software are being actively encouraged to adopt an OSS development strategy for developing their products (Hecker, 1999). Adherents of OSS argue categorically that OSS should, and more importantly will, become the dominant form of software production (Raymond, 1999).

Popular accounts of open-source development convey the “bazaar” notion of software development where there is an open (and seemingly democratic) discussion of changes to ensure that the best changes are accepted and that new designs evolve organically through public interaction with a large number of people making contributions (Feller and Fitzgerald, 2002). In this paper we step back from the emotional aspects of this issue and try to address a question, the answer to which has been taken for granted without much actual evidence: do open-source
projects differ from commercial projects in terms of the distribution of labor in the software development and maintenance process?

The following questions are addressed in this paper: What is the distribution of overall contributions to the OSS software products and how does it compare to similar commercial projects? What is the distribution of contributions of the developers across various types of software development activities (feature development, bug-fixing, configuration management and documentation)? What are the plausible explanations for overall contribution patterns based on available skills and software development environments?

Our research is fundamental to understanding the competitive positions of OSS versus traditional commercial software development. The claim that OSS has inherent advantages over traditional development is based in large part on a belief that OSS development is in fact very different. If the development practices are in fact similar, then any differences between the two development models are severely attenuated.¹

We conduct an empirical analysis of the development of Apache and FreeBSD OSS projects based on their respective source code change logs. Identifying the types of changes made to the source code as well as identifying the person(s) contributing to these changes, we contrast our findings from the OSS projects with four similar commercial software development projects. We proscribe a metric to measure the distributed nature of contributions and develop an analytical basis to explain aggregate contribution patterns observed.

¹ If the general models are the same, then the advantages of one versus the other depend on factors such as the relative costs of workers in the two models (an interesting issue that we examine in detail elsewhere) and the relative quality of workers available under the two systems.
Empirically we find that, similar to commercial projects, the distribution of development in OSS projects is concentrated in the hands of a few developers. These findings show that OSS projects are not significantly different from commercial projects in terms of their labor distribution, despite their frequent characterization as more like a freewheeling ‘bazaar’ than the controlled commercial development process referred to as a ‘cathedral’ (Raymond, 1999). This finding has significant ramifications, not only for OSS projects, but also for firms looking to invest in, hire personnel from or adopt the development form of OSS projects.

In the following section we outline the relevant literature on software development contribution patterns, programmer productivity and motivations of OSS programmers. In Section 3, we briefly describe the source for our data, and the terms and measures used to quantify and compare the contribution distributions in the OSS and commercial software projects. Section 4 describes our empirical findings while section 5 provides the analytical basis to explain the empirical observations and a discussion of the results. Section 6 concludes by discussing the implications of our findings and posing some questions that remain to be investigated.

2. RELEVANT RESEARCH BACKGROUND

OSS has received considerable attention from the popular press, industry, and academia. The history and development of OSS is described by DiBona et. al. (1999). Raymond (1999) and Feller and Fitzgerald (2002) describe the OSS process of active collaboration amongst loosely structured developers located all over the world. In a description of the Apache server project by one of the lead contributors, Fielding (1999) describes a voting cum minimal quorum consensus mechanism to approve new ideas or include new changes to the source-code. He also states that the project is a meritocracy where developers contributing to the project are given more powers
to make changes. With many distributed developers collaborating and contributing to the source code, it seems reasonable to expect that development in an OSS project is democratic (assuming appropriate skill, there is no barrier to contributions) hence somewhat evenly distributed amongst developers, with new features emerging out of this global collaboration.

With respect to the motivations of the OSS developers, Raymond (1999) suggests that most developers are also users of the software product who make changes in part to suit their own needs, or rather “to scratch their own itch”. He attributes additional reason for developers to make contributions to projects as a form of altruism or more specifically a “post scarcity gift culture”. On the contrary, Lerner and Tirole (2002) in their examination of OSS projects suggest that developers have the incentives of “career concerns” or “ego gratification” which they classify as “signaling incentives”. The authors further assert that the OSS developers are strongly motivated to get better employment or venture funding for a new enterprise even if the payoff comes with a lengthy delay.

Brooks (1995) in his essay on software development acknowledges the wide productivity variations amongst programmers and asserts the need for relatively large teams to build large systems on a meaningful schedule. He echoes Mills’ (1971) idea that a development team be structured like that of a surgical team – where the ‘surgeon’ or ‘chief programmer’ and his ‘co-pilot’ ensure conceptual integrity of the system through the architecture design. He further suggests that teams should be organized hierarchically, while Raymond (1999) suggests that this ‘cathedral’ or top-down structure is not relevant in an OSS setting. It should however be noted

______________

2 This signaling result has been further analyzed and empirically validated by Hann et. al. (2002)
that Brook’s projects were new developments while most current OSS projects may be characterized as maintenance projects since the products have already been ‘delivered’ and the current actions on the products are mostly modifications to correct faults, improve performance, or adapt to a changed environment (Lientz and Swanson, 1980, ANSI/IEEE 1983).

Boehm (1981) in the development of his cost model discusses the wide range of productivity observed in earlier studies amongst commercial software developers and agrees to the view that bulk of the productivity in projects comes from a relatively small number of developers. Given the hierarchical nature of commercial projects, one would expect this concentration as the more productive developers would have responsibility, control and authority over a larger segment of the project.

Prior literatures on OSS attempt to highlight how OSS projects differ from traditional commercial development. Feller and Fitzgerald (2002, pg84) describe the characteristics of a generic OSS development process as involving parallel development of multiple solutions to a problem which compete with each other by a large community of distributed developers. Raymond (2000, pg41) posits that the bugs in an OSS product “turn shallow quickly when exposed to a thousand eager co-developers.” The image one gathers from this literature is that OSS projects have a large, distributed developer base and that software development (and bug fixing) is occurring through a competitive environment. The essence of “shallow bugs” implies that there are indeed many co-developers that are sifting through submitted source-code, spotting errors and fixing them. Given a large developer base, an even distribution of output amongst developers would be expected.

However, Mockus et. al.(2002), show that 83% of the modifications requests in the Apache
server project are made by the top 15 developers (or 3% of all contributors). In a similar type of study, Dinh-Trong and Bieman (2005) show that in the FreeBSD project 80% of the changes are contributed by the top 50 developers. In other studies of OSS projects, Ghosh and Prakash’s (2000) survey of contributions to free software show that the top 10% of the authors contributed to 72% of the code base. In a study of the top 100 mature OSS projects hosted on SourceForge, Krishnamurthy (2002) finds that the median number of developers in the OSS-SourceForge projects was 4. This is considered to be a relatively small number and the concentration of development is surprising since we have been led to believe that there are “thousands of eager co-developers pounding on every new release” (Raymond 1999, pp41).

It is not clear from the earlier empirical studies whether the concentration of contributions in OSS projects is different from that observed in commercial projects. It is also not clear if this pattern holds for all types of development such as design, bug-fixing, configuration management, documentation and bug-reporting. The focus of this paper is to fill that lacuna. We also develop an analytical basis to explain the observations of relative concentration of development.

3. DATA DESCRIPTION AND MEASURES

3.1 The Apache Web Server Project

The genesis of the Apache web server was a public domain web server developed at the National Center for Supercomputing Applications (NCSA), called Mosaic. The original server

Note that these numbers are aggregated for all changes and the specific types of contributions and the actual contributors are not reported.

SourceForge is an online repository of a number of OSS projects. http://sourceforge.net/

The Apache HTTP Server Project can be accessed at: http://httpd.apache.org/
was maintained and frequently patched by a number of independent non-commercial developers. In spite of the informal and non-commercial nature of the Apache server project, a recent Netcraft survey\(^6\) (November 2005) shows that it runs 71% of the total websites. In methodology similar to that described in Mockus et. al. (2002), we gather our data from publicly available information in the distribution list archives of the Apache server project. The time period of our data is from February 1995 through December 2000.

3.2 The FreeBSD Project

The FreeBSD project\(^7\) is an operating system derived from the BSD Unix developed at the University of California, Berkeley. FreeBSD is widely used. According to Netcraft (June 2004) FreeBSD hosted almost 2.5 million active web sites with nearly 5 million host names\(^8\). The FreeBSD development process is defined and well documented. Detailed information concerning the internal workings of the FreeBSD project is readily available through the e-mail archives, bug database, and the source code repository. In this study we gather our data from the FreeBSD source code repository over the 1993 to 2005 time period.

3.3 The Commercial Projects

For the commercial projects we use available data from Mockus et. al. (2002). The projects labeled B, C, D and E are telecommunications related projects with approximately the same size.


\(^{7}\) http://www.freebsd.org/

\(^{8}\) Source: Netscraft news: http://news.netcraft.com/fullindex
as the Apache server project\textsuperscript{9}. Project B involved call handling software for a wireless network, while C, D and E were operations administration and maintenance support software for telecom products. The data we have for these projects is the aggregate contributions of the various developers - hence the level of analysis possible for these projects is somewhat limited.

\textbf{3.4 Description of Terms: Apache and FreeBSD}

As with any OSS project, both the object-code as well as the source code is publicly available. The central repository for the source code in the case of both the Apache and FreeBSD projects was an open-source version control system for its source code called Concurrent Version Systems (CVS)\textsuperscript{10}.

Version control software typically stores text files and multiple versions of the same. They allow users to download the files and keep track of changes made to them. They provide secure communication and have other administrative functions that make such version control systems ideal for storing and managing the source code of any software project. Every time a developer (who has permission) uploads changed files, the transaction is called a ‘commit’. A single commit could consist of a number of lines of code distributed amongst a number of files. For example, a developer, in order to fix a bug makes changes to files main.c, main.h, interface.c and interface.h. These files are then uploaded to the central repository as one commit transaction. In addition the developer also fills in documentation regarding the changes made and other developers that have assisted him in the activity. The ‘commit’ transaction constitutes a

\textsuperscript{9} Projects D and E are said to be smaller in size than projects B and C
\textsuperscript{10} Details of CVS can be obtained at http://www.nongnu.org/cvs/
modification request (MR) and is logged in the version control system. We consider the MR synonymous to a unit of work.

From the logs of all commit transactions one can obtain additional information about who submitted the change or who contributed to the change. The sources are attributed in the fields “Submitted by” or “Obtained from”. OSS developers are expected to attribute contributions and give credit where it is due (Raymond, 1999). Hence, if these fields are blank then the person who commits the change is considered the primary contributor. Thus, we obtain information of all the MRs, the type of contribution and their respective contributor(s).

3.5 The Gini Coefficient as a Measure of Concentration

The Lorenz curve (Lorenz, 1905) is commonly used to represent income distribution in economies. It has also been used in other domains to represent inequality in size of a biological species population (Damgaard and Weiner, 2000). The Lorenz curve is a plot of cumulative fraction of contribution to the cumulative fraction of participants ordered by size. Given a sample of n ordered individuals with $x_i$ the size of individual i and $x_1 < x_2 < x_3 \ldots < x_n$, then the sample Lorenz curve is the polygon joining the points $(h/n, L_h/L_n)$ where $h = 0,1,2\ldots n$, $L_0=0$, and

$$L_h = \sum_{i=1}^{h} x_i$$

Alternatively, the Lorenz curve, $L(y)$, can be expressed as:

$$L(y) = \frac{\int_{0}^{y} x dF(x)}{\mu}$$

where $F(x)$ is the cumulative distribution function of ordered individuals, $\mu$ is the average size of contribution and $0 = y = 1$. See Figure 1 for an example of a Lorenz curve.
A measure used to summarize the Lorenz curve is the Gini coefficient (Dixon et. al., 1987 and Dixon et. al., 1988). The Gini coefficient is the area between a Lorenz curve and the diagonal which represents perfectly even distribution (See Figure 1). The Gini coefficient ranges from a minimum value of zero, when all individuals have equal contribution to the total, to a theoretical maximum of one in an infinite population in which every individual except one has a contribution of zero. The advantage of the Gini coefficient as a measure is that it can be used to compare across different sectors independent of the number of observations. In order to compare across various software projects, we use the Gini coefficient to measure the inequality in contributions across the developer population. The Gini coefficient is mathematically expressed as: 

\[ G = 1 - 2 \int_0^1 L(y) \, dy. \]

For a discrete population ordered by increasing contribution of the individual elements, it can be computed as:

\[ G = \frac{\sum_{i=1}^{n} (2i - n - 1)x_i}{n^2 \mu}. \]

We use the latter equation to compute the Gini coefficient for our data.
Another index traditionally used in economics to measure market concentration is the Herfindahl Hirschman Index (or HHI). This index may also be used in the OSS context to represent the concentration of development. A higher value of HHI indicates a higher concentration of development (i.e. few developers doing most of the work). However, a limitation of the HHI is that it is sensitive to the size of the population under examination. Larger populations will exhibit lower HHI even if they are relatively concentrated. One of the advantages of the Gini coefficient as a measure is that it is relatively insensitive to the size of the population. Given that the software projects we are comparing have different numbers of developers, the Gini coefficient is a better measure than HHI for the distribution of the contributions in a software project.

4. EMPIRICAL RESULTS

4.1 The Distribution of Total Software Development Effort

In this section we describe our empirical observations and measures. Our initial analysis of the modification logs confirms the observations of Mockus et. al. (2002) and Dinh-Trong and Bieman (2005), showing the concentrated distribution of development for the OSS projects. We also observe a similar pattern in the commercial projects. The Lorenz curves for all the projects are shown in Figure 2 and Table 1 records the respective Gini coefficients. We see that the Gini coefficient for effort in the OSS projects is higher than that observed in the commercial projects. This implies that the concentration of effort is greater in OSS projects compared to commercial projects.
We must note that these MRs aggregate all types of effort, namely bug-fixing, feature enhancements, configuration management and documentation. In the following section we further examine and measure the distribution of contributions within the OSS projects across various types of project activities, namely, feature development or bug-fixing, configuration management and documentation.

**Table 1: Gini Coefficient for Aggregate Contributions**

<table>
<thead>
<tr>
<th>OSS Projects</th>
<th>Gini Coefficient</th>
<th>Commercial Projects</th>
<th>Gini Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Server</td>
<td>0.912</td>
<td>Project B</td>
<td>0.49</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>0.837</td>
<td>Project C</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project D</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project E</td>
<td>0.66</td>
</tr>
</tbody>
</table>
4.2 Distribution of Effort by Type of Change

The Lorenz curves for the total contributions to the various projects are shown in Figure 2. We further break down the contributions to the projects into coding (identified by changes to source code files), configuration management (identified by changes to configuration and script files) and documentation (identified by changes to documentation text and html files). Figure 3 shows the Lorenz curves for the Code, Documentation and Configuration Management files for the OSS projects. We observe that the pattern of concentration is also reflected in these types of contributions. The Gini coefficient values for the contributions by type are shown in Table 2. We can see from the figures and values that the concentration of effort is relatively high (G>>0.5). The Gini coefficient values for configuration management (amongst Apache and FreeBSD) are relatively similar (difference of 0.01) compared to those of Coding and Documentation (difference of ~0.1). We conjecture that the since the code base of FreeBSD is much larger and varied than Apache, contributions are likely to be more distributed, whereas the configuration management contributions are limited irrespective of the size of the software.

**Table 2: Gini Coefficient values for various types of contributions**

<table>
<thead>
<tr>
<th>Contribution Type</th>
<th>Apache Server Project</th>
<th>FreeBSD Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>0.916</td>
<td>0.810</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>0.832</td>
<td>0.844</td>
</tr>
<tr>
<td>Documentation</td>
<td>0.907</td>
<td>0.834</td>
</tr>
</tbody>
</table>
4.3 Analysis of Bug Fixes

One type of contribution that we consider to be different from those described earlier is that of bug reporting. Bug reporting is relatively easy or even automated; it imposes a low threshold of energy and talent: it only requires the user to identify the existence of a problem, not its source. Thus, while reporting is likely to be more distributed it is not clear what the pattern for fixing these bugs should be. From the GNAT database of the Apache project we examined the
contribution of bug reports and bug fixes. In our analysis we find a very interesting distinction in this activity of the project. While the number of bug reporters in the time period analyzed was relatively large (5346), the total number of individuals that were involved in fixing of bugs was comparatively small (49). Our data shows that though the involvement in bug reporting is close to uniform ($G = 0.21$), the involvement of the community in fixing of those bugs is still concentrated ($G = 0.82$) although slightly less concentrated than the overall contributions for Apache. The Lorenz curves for the bug reporting and bug resolution distribution are shown in Figure 4. We thus see that the popular impressions of widely distributed development contributions is correct for bug reporting but not for the fixing of those bugs.

**Figure 4: Lorenz curves for Reporting and Resolution of bugs in Apache**
Table 3: Gini Coefficient for Bug Reporting and Resolution Contributions

<table>
<thead>
<tr>
<th>Contribution Type</th>
<th>Apache Server Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug reporting</td>
<td>0.210</td>
</tr>
<tr>
<td>Bug resolving</td>
<td>0.825</td>
</tr>
</tbody>
</table>

We conclude from these observations that the concentration of contribution to OSS projects is higher than the concentration of development in commercial projects. This pattern of high concentration of overall contributions (Gini coefficient values are greater than 0.8) is also observed for various sub-types of contributions such as coding, documentation, configuration management and bug fixing. It is only for bug reporting that there is low concentration.

The question that this immediately raises is: Is this result unexpected? Is it at all consistent with the description of OSS development? Assuming that there are thousands of developers sifting through the code and making contributions, could we nevertheless observe this level of concentrated development? Under what conditions, if any, would we observe these results? In the next section we develop some analytical methods to examine this. We examine various assumptions and the results based on these assumptions and lend some intuitions regarding this issue.

5. ANALYTICAL MODELS

In the previous section we observed the distributional pattern of individual developers making contributions. In this section we examine possible micro underpinnings of the software development environment and the resultant impacts on the distributional patterns.
A software product can be considered to be the cumulative effect of a large number of atomic (programming) work tasks \(N_w\). Each work task can be considered equivalent to a delta or a modification request (MR) as described by Mockus et. al.(2003). For the sake of simplicity, in our analysis we assume that each work task has the same level of complexity. In this analysis we only consider a population of developers \(N_d\) that possess the expertise to accomplish a work task. Expertise is defined here as the specialized skill, knowledge and training that an individual possesses that would allow her to successfully perform a (programming) work task. Let capacity be the primary factor that determines the level of output of an individual developer. A high capacity developer is one that typically has a high output. The cumulative contribution distribution of any software project thus depends on two factors: The distribution of developer capacity in a population of developers and the relative output level of the developers by capacity\(^{11}\). We now examine these two factors in more detail.

**5.1 The distribution of developer capacity**

We defined capacity as a factor that determines level of output of a developer. It reflects the combination of the expertise that a developer possesses as well as the amount of time she has at her disposal to work on the problem. While individual ability is often considered and modeled to be normally distributed (Rothschild and Stiglitz, 1982), it is not clear what the distribution of capacity will be. The population we are considering in our analysis is developers that have the expertise to perform the programming tasks and is not the same as the general population of users. For our analysis we consider two well known distributions \(f(x)\) that might be thought to

\(^{11}\) Essentially, output as a function of the individual capacity
reasonably capture the distribution of capacity in a developer population. (Figure 5)

![Figure 5: PDF of capacity](image)

**Uniform density:** In this case we consider the situation where the distribution of developers’ capacity in the chosen population is uniform. The assumption here is that the fraction of developers is the same regardless of their capacity, i.e. there are as many high capacity developers as there are low. By normalization, \( f(x) = 1 \) for \( x \in [0,1] \).

**Decreasing density of capacity:** Here we consider the situation where the fraction of total developers decreases with increase in capacity. This assumption takes into account the fact that there are fewer high capacity developers than low capacity developers. A functional form that is appropriate for this assumption is the exponential distribution \( f(x) = \lambda e^{-\lambda x} \) and it satisfies the normalization condition for density functions: \( \int_{0}^{\infty} f(x).dx = 1 \).

As noted earlier, capacity reflects the expertise level as well as the time available. In commercial development environments, developers of similar expertise levels are hired and conditions of employment require that developers spend a similar number of hours at work. The effort of
employed developers is generally monitored by the firm even if only imperfectly. Thus a uniform and truncated (narrow) distribution of capacity might well be considered suitable for this environment. On the other hand, in OSS projects, expertise as well as the available time of individual developers could vary considerably\(^{12}\). There are undoubtedly fewer high expertise developers than low expertise developers. It also seems reasonable to expect that there will be still fewer high expertise developers with large amounts of time they can devote to OSS development since the opportunity cost of time will be very high for these individuals. Hence the decreasing density function of capacity would seem to reasonably represent developers in OSS projects.

5.2 Output level of developers

The output one can observe in an OSS project is the number of tasks accomplished by an individual developer. As we mentioned earlier, Lerner and Tirole (2002) describe career concerns and reputation as motivations for OSS programmers. Hence, for a developer to maximize her utility from the perspective of career concerns, it is natural to presume that she will attempt to maximize the number of quality adjusted\(^{13}\) tasks accomplished. As support for this view, Fielding (1999) reports that OSS reputation is positively correlated to the number of work tasks an individual accomplishes.

If the total number of tasks (or output) accomplished by an individual developer is \(t_i\), then we can represent the relationship between output, \(t_i\) and capacity \(x_i\) very simply as \(t_i = t(x_i)\), where

\(^{12}\)Boehm (1981) reports that the productivity ratio of top to bottom programmers can be as large as 26:1

\(^{13}\)Quality adjusted in the sense of difficulty and the impact on reputation
Asundi et al.: Understanding OSS Contributions

\[ t(x+e) = t(x)^{14} \]. A linear relationship between the developer’s capacity and output is a natural subset of a more general relationship. However, we note that in commercial organizations tasks are allocated to developers, but in OSS projects there is no strict coordination (due to loosely coupled development) and hence the more productive developers will pre-empt the less productive developers and perform most of the tasks. For this reason we examine different functional forms for \( t(x) \), their impact on the Lorenz curves and their corresponding Gini coefficients. This provides an interesting and likely explanation for the phenomenon we reported in Section 4.

5.3 Analysis

If \( f(x) \) is the density function of developers by capacity \( x \) and \( t(x) \) is the number of tasks a developer of capacity \( x \) accomplishes, then the fraction of total development accomplished by the developers up to capacity \( m \) can be expressed as:

\[
L = \frac{\int_0^m t(x) f(x) \, dx}{\int_0^\infty t(x) f(x) \, dx}.
\]

The fraction of developers that accomplishes this fraction of total development is expressed as:

\[
y = \int_0^m f(x) \, dx.
\]

For the case where we consider the density function to be uniform, the fraction of developers up to capacity \( m \) is \( y=m \). Similarly, for the exponential case, the fraction of developers up to capacity \( m \), is

\[
y = 1 - e^{-\lambda m}.
\]

The Lorenz curve is the fraction of total development, \( L \) expressed as a function of the fraction of total developers responsible for the development, \( y \). We now consider the

\[ 14 \text{ We assume that the function is monotonically increasing.} \]
following broad cases for the functional forms for $t(x)$ and compute the respective Lorenz curves and Gini coefficients:

**Case 0:** We first consider what we might refer to as the ‘thousand eyeball’ case where we assume that there are a large but finite number of tasks and a very large number of developers ($N_w<<N_d$). In this case, the number of high capacity developers, although a small portion of the total distribution of developers, is nevertheless large relative to the number of tasks. After each developer picks an independent task and attempts to solve it\textsuperscript{15}, we will find a fairly equitable distribution of completed tasks because the high capacity developers, who are similar to one another in terms of capacity, will be able to solve most of the tasks. This leads to a fairly even distribution of completed tasks for the most part, although there will be some tasks completed by low capacity developers. This is particularly true because the lack of central coordination will mean the high capacity developers will choose, by happenstance, the same the tasks as low capacity developers but the high capacity developers are far more likely to finish them. The efforts of low capacity developers will largely go to waste since they will be negated by the faster completion of the same tasks by high capacity developers.

The Lorenz curve in this case would approach the diagonal line and the Gini coefficient would not stray too far from $G=0$. Clearly, we do not observe this situation in reality. Hence we need to examine cases where the number of tasks is large but finite and the number of developers is relatively smaller ($N_w>>N_d$). Indeed, if one examines the two OSS projects (Apache and FreeBSD), we observe that there are approximately 50 to 100 developers and 1000 to 4000

\textsuperscript{15} A situation that Raymond (1999) describes as “thousands of eager co-developers pounding on every new release”
unresolved bugs, which is an indicator of the total number of tasks in the projects.

**Case 1:** \( t(x) = t^* \) (a constant). In this case we assume developers irrespective of their capacity accomplish the same number of tasks. This situation is plausible in an extreme case where the high and low capacity developers accomplish a fixed number of tasks (and not more) in order to be visible as developers. Since every developer completes the same number of tasks, it is obvious that the Lorenz curve follows the diagonal the Gini coefficient, \( G = 0 \).

**Case 2:** \( t(x) = k \cdot x \). A seemingly reasonable assumption is that the output of a developer would be proportional to or a linear function of her capacity. This case assumes that with increasing individual capacity, the number of tasks accomplished will increase proportionally. It is shown in the Appendix that for the uniform density function, the Gini coefficient, \( G = 0.333 \) and for the exponential density function, the Gini coefficient, \( G = 0.5 \). Thus for this linear relationship between capacity and output the Gini coefficient takes values less than or equal to 0.5 which is lower than our measurements for both commercial and OSS projects.

![Figure 6: t(x) case 1 and 2](image)
Case 3: \( t(x) = t^* (1 - e^{-\eta x}) \). In this case we consider the situation where the number of tasks accomplished by developers with a low capacity is small, but as the capacity increases, the output increases dramatically and asymptotically to a value \( t^* \) (see Figure 7). This functional form for \( t(x) \) could exist in commercial development organizations where the output level of developers stops increasing because developers ‘satisfice’ instead of maximize, meaning they reduce their efforts once they reach a comfortable level of output, which might occur if rewards are not strongly related to output productivity.

High values of \( \eta \) imply that the level of output asymptotes quickly with increase in capacity. For the uniform density assumption, large values of \( \eta \), \( t(x) \) tends to \( t^* \) the constant case and \( G \approx 0 \). As \( \eta \) gets smaller\( ^{16} \), \( t(x) \) becomes similar to the \( k.x \) case and for \( \eta=1 \), \( G=0.33 \). For the exponential density case, \( G \) tends to 0.5 (See Appendix for calculations). Clearly this case does not reflect our observations from Section 4 either.

Case 4: \( t(x) = k.e^{0.x} \) Finally, we consider the case where task output of the developers is an exponential function of their respective capacity. This case reflects a low output for lower capacity developers and while level of output increases slowly with capacity, the task output for high capacity developers is exponentially higher. (See Appendix for detailed calculations).

\[ ^{16} \text{There exists a lower bound on } \eta \text{ since } N_D \int_0^1 t(x).dx = N_w \text{ and hence can never be close to zero as } Nw >> Nd. \]
a) For the uniform density distribution and for small values of $\theta$, $G = 0$ whereas for reasonably large values\(^{17}\) of $\theta(=5)$, $G \approx 0.6$. b) For the case of decreasing density distribution, the Gini coefficient ranges from 0 to 1.

**5.4 Discussion of results**

In our analysis we have covered a spectrum of possibilities regarding the distribution of capacity (uniform and decreasing density) as well as the output functional forms (constant, proportional, inverse-exponential and exponential) although we have by no means exhausted all possibilities. We observe that the Gini coefficient value for various assumptions in Case 1 through Case 3, varies from 0 to 0.5 (see Table 4 for a summary of the results).

Our empirical findings (from Section 4) show that the Gini coefficients of both the OSS projects are considerably greater than 0.5 which is only compatible with Case 4. The commercial projects, in comparison, have Gini coefficients close to 0.5 and the environment may be explained by either Case 2, 3 (with a decreasing density function) or Case 4.

The difference between commercial and OSS contribution distributions can be explained by the difference in their respective environments (dependent on developer expertise and available time). While expertise will vary in both environments (more so in OSS projects), time spent on work is relatively invariant amongst developers in commercial environments due to project management practices of regulated work hours, activity reports etc. In OSS environments, on the

\[^{17}\] Here, there is an upper bound on $\theta$ since $\int_0^1 t(x) dx = N_w$ and hence can never be very large.
other hand, time spent on work is unregulated and has a high variance given that most developers rarely work full time on an OSS project.

Table 4: Gini coefficient ranges for various model assumptions

<table>
<thead>
<tr>
<th>Functional form for ( t(x) )</th>
<th>Uniform Density</th>
<th>Decreasing Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1. ( t(x) = t^* )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Case 2. ( t(x) = kx )</td>
<td>0.333</td>
<td>0.5</td>
</tr>
<tr>
<td>Case 3. ( t(x) = t^* (1 - e^{-\eta x}) )</td>
<td>[0,0.33]</td>
<td>[0,0.5)</td>
</tr>
<tr>
<td>Case 4. ( t(x) = k e^{0.6x} )</td>
<td>[0,0.6]</td>
<td>[0,1]</td>
</tr>
</tbody>
</table>

The only case that can yield very high Gini coefficient values is that of Case 4 with decreasing density function for capacity. The assumptions of Case 4b imply that there are a few very high capacity developers who are exponentially more productive than the lower capacity developers. This, then, might represent the underlying conditions actually in the market.

While the reasons for this disparity in capacity and outputs are not completely clear, a plausible explanation could be the paucity of available time for the high expertise OSS developers (because such developers normally have a high opportunity cost for their time), leaving only a small number of talented developers with a great deal of available time to accomplish a disproportionately larger fraction of the total tasks. Another possible explanation is that the task complexity could be such that the high capacity developers are those with an unusually high level of expertise who find the tasks relatively easy to work on in the given time and thus accomplish a disproportionately large fraction of them. If, for whatever reason, those with the greatest expertise also had the most time, we would also find such a result.
From a software development perspective there are good reasons to believe that there is significant heterogeneity of developer capacity to explain the more concentrated output in OSS projects. This analysis is but a first attempt at explaining this phenomenon of highly concentrated development and determining a more complete explanation is a question for future research.

6. CONCLUSION AND IMPLICATIONS

“A lot of security problems derive from the core, (with open-source code), thousands of people look at the critical portions of source code and check those portions are right. It's a major advantage to have open-source code.”

Bertrand Serlet, Senior Vice-President of Software, Apple.18

The notion expressed in the above quote, that there are thousands of capable developers sifting through the code in OSS development, was the focus of this paper. We empirically examined, through an analysis of source code archival logs, the contribution patterns of OSS and commercial projects.

It appears that the nature of software development is largely universal irrespective of the type of software development methodology being adopted; it requires capable and productive developers and development is not distributed evenly amongst its developers. Bug reporting imposes a low threshold of energy and expertise, requiring only the identification of a problem, not its source. Fixing problems is an entirely different story, with few individuals having the inclination or the intricate knowledge of the source code and the software architecture to successfully engage in

Using the Gini coefficient as a metric to measure the extent of concentration of contributions in a software project we found that the aggregate contribution patterns for the development of the OSS projects are concentrated in the hands of a small fraction of developers—more so than for commercial projects. This concentrated pattern of OSS contribution is also observed for detailed types of changes such as code development, documentation and configuration management. This result is the exact opposite of the mythology surrounding the ‘thousand eyeball’ descriptions of OSS described in the above quote.

In an attempt to understand how these results might come about, we developed analytical models to see which distributions of developer characteristics might be capable of explaining the aggregate contribution phenomena found in these software development projects. The contribution patterns empirically observed are plausible only when there are few high capacity developers accomplishing a disproportionately large number of tasks for the project.

A limitation of our empirical analysis is that it was based on only two OSS projects and four commercial projects. Nevertheless, the Apache server and the FreeBSD project are presumably representative of other successful OSS projects. All the same we acknowledge the need for similar analyses on other OSS projects to validate the generality of our results.

The implication of our findings for organizations that adopt OSS products is that they should not expect to be entering an entirely different development universe. For firms looking to adopt the OSS methodology by releasing their products to the community, the implication of this research
is that they must continue to invest the majority of the effort in the various software development and maintenance activities or to nurture a small base of friendly developers who will do the lion’s share of the work. They must understand that with increasing complexity of software tasks, the probability of finding community developers with the requisite capabilities diminishes.

Can OSS remain truly open? Has it ever really been that way? Does it even matter? Our findings show that the control of the future direction of the OSS projects rests in the hands of very few individuals. With respect to project development the differences between OSS and commercial software are far smaller than normally understood. With the increased participation of commercial organizations in OSS projects, there is a possibility that this direction may be guided by a particular organization’s commercial goals and that these distinctions will be further blurred. Nevertheless, given the nature of most OSS licenses, the source code will remain open for all to manipulate. Is there an advantage of one system over the other? We look forward to future research to help us answer these questions.

7. REFERENCES


8. APPENDIX

Presented here are some of the mathematical calculations for the Lorenz curves and Gini coefficients for the various cases.

**Case 2: \( t(x) = k \cdot x \)**

For the uniform density assumption, the Lorenz curve, \( L(y) = \frac{\int_0^y x \, dx}{\int_0^1 x \, dx} = y^2 \) and the Gini coefficient \( G = 1 - 2 \int_0^1 y^2 \, dy = \frac{1}{3} \).

For the decreasing density function, \( f(x) = \lambda e^{-\lambda x} \), \( m = \ln(1 - y)^{-1/\lambda} \), the Lorenz curve is:

\[
L(y) = \frac{\int_0^m x e^{-\lambda x} \, dx}{\int_0^m x e^{-\lambda x} \, dx} = y + (1 - y) \ln(1 - y) \text{ and}
\]


the Gini coefficient \( G = 1 - 2 \int_0^\infty (y + (1 - y) \ln(1 - y))dy = \frac{1}{2} = 0.5 \).

**Case 3:** \( t(x) = t_*(1 - e^{-\eta x}) \)

For the uniform density assumption, the Lorenz curve \( L(y) = \frac{\int_0^y (1 - e^{-\eta x})dx}{\int_0^\infty (1 - e^{-\eta x})dx} = \frac{\eta_y - e^{-\eta y} - 1}{\eta + e^{-\eta} - 1} \) and the Gini coefficient is: \( G = 1 - 2 \int_0^\frac{1}{\eta} \frac{\eta_y - e^{-\eta y} - 1}{\eta + e^{-\eta} - 1}dy = \frac{e^{-\eta}(\eta + 2) - 3\eta + 2}{\eta(\eta + e^{-\eta} - 1)} \).

For the decreasing density function, \( f(x) = \lambda e^{-\lambda x}, \ m = \ln(1 - y) \frac{1}{\lambda} \)

the Lorenz curve \( L(y) = \frac{\int_0^m (1 - e^{-\eta x})e^{-\lambda x}dx}{\int_0^\infty (1 - e^{-\eta x})e^{-\lambda x}dx} = \frac{\lambda}{\eta} \frac{1 - y^{\frac{\eta + \lambda}{\lambda}}}{1 - y^{\frac{\eta + \lambda}{\lambda}}} = \frac{\eta + \lambda}{\lambda} (1 - y) + 1 \) and

the Gini coefficient \( G = 1 - 2 \int_0^\frac{\lambda}{\eta} \frac{(1 - y)^{\frac{\eta + \lambda}{\lambda}} - \frac{\eta + \lambda}{\lambda} (1 - y) + 1}dy = \frac{\lambda}{\eta + 2\lambda} \). Hence for small values if \( \eta \), \( G \) tends to 0.5.

**Case 4:** \( t(x) = k e^{0.2x} \)
For the uniform density assumption, the Lorenz curve
\[
L(y) = \frac{\int_0^y e^{\theta \cdot x} \, dx}{\int_0^1 e^{\theta \cdot x} \, dx} = \frac{e^{\theta \cdot y} - 1}{e^\theta - 1}
\]
and

the Gini coefficient
\[
G = 1 - 2 \int_0^1 \frac{e^{\theta \cdot y} - 1}{e^\theta - 1} = \frac{\theta - 2}{\theta} - \frac{2}{e^\theta - 1}.
\]

For the decreasing density function, \(f(x) = \lambda e^{-\lambda \cdot x}\), \(m = \ln(1 - y)\)
\[
\int_0^m e^{\theta \cdot x} \cdot e^{-\lambda \cdot x} \, dx
\]
the Lorenz curve is
\[
L(y) = \frac{\int_0^y e^{\theta \cdot x} \cdot e^{-\lambda \cdot x} \, dx}{\int_0^\infty e^{\theta \cdot x} \cdot e^{-\lambda \cdot x} \, dx} = 1 - (1 - y)^{(1-\gamma)}
\]
and

the Gini coefficient
\[
G = 1 - 2 \int_0^\infty 1 - (1 - y)^{(1-\gamma)} \, dy = \frac{\theta}{2\lambda - \theta}.
\]
For reasonable values of \(\theta\) (when \(0 \leq \theta \leq \lambda\)), \(G\) can take any value between 0 and 1.